NIST Handbook 160

Nanolithography Toolbox


This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.HB.160
Chapter Contents

- Platform Independent
- Vertex control of shapes
- Curved geometries
- Polar and Hexagonal Arrays
- Pillar - Hole Arrays
- Grating Couplers
- Spirals
- Gratings
- Verniers
- Arbitrary Function Generator
- Label Maker - Text To GDS
- Fractals
- Grayscale Image to GDS
- Scripting
- CNST Nanophotonics Library
- Couplers
- Photonic Crystals
- Waveguides, S-Bends, Y-Bends
- Custom Tapers
- Reticle Frame Generator
- JEOL Job File Generator
- JEOL Write Time Estimation
# Contents

1 Nanolithography Toolbox

## Nanolithography Toolbox

1.1 Overview

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1 Terms Of Use</td>
<td>8</td>
</tr>
<tr>
<td>1.1.2 Disclaimer</td>
<td>10</td>
</tr>
<tr>
<td>1.1.3 Acknowledgements</td>
<td>10</td>
</tr>
<tr>
<td>1.1.4 Development Team</td>
<td>10</td>
</tr>
<tr>
<td>1.1.5 Future Developments</td>
<td>11</td>
</tr>
<tr>
<td>1.1.6 Java 8 Requirements</td>
<td>12</td>
</tr>
<tr>
<td>1.1.7 Distribution Package</td>
<td>12</td>
</tr>
<tr>
<td>1.1.8 Modes of Operation</td>
<td>12</td>
</tr>
<tr>
<td>1.1.8.1 CNST Scripting</td>
<td>12</td>
</tr>
<tr>
<td>1.1.8.2 Graphical User Interface</td>
<td>15</td>
</tr>
</tbody>
</table>

## GDS Scripting

2.1 Setup, Execution and Examples

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.1 Quick Start Setup</td>
<td>16</td>
</tr>
<tr>
<td>2.1.2 Setup - CNSTdefaultValues.xml open and save directories</td>
<td>17</td>
</tr>
<tr>
<td>2.1.3 Graphical User Interface Execution</td>
<td>18</td>
</tr>
<tr>
<td>2.1.4 Running CNST Scripts From a Terminal Command Prompt</td>
<td>19</td>
</tr>
<tr>
<td>2.1.5 Running Multiple CNST Scripts - Batch File Mode</td>
<td>19</td>
</tr>
<tr>
<td>2.1.6 Running CNST Scripts Within Matlab</td>
<td>20</td>
</tr>
<tr>
<td>2.1.7 Python Script Generation</td>
<td>21</td>
</tr>
<tr>
<td>2.1.8 NotePad++ Syntax Coloring</td>
<td>22</td>
</tr>
<tr>
<td>2.1.9 Scripting Examples</td>
<td>23</td>
</tr>
<tr>
<td>2.1.9.1 Basic Scripting Example</td>
<td>23</td>
</tr>
<tr>
<td>2.1.9.2 Scripting Example - Labeled Calibration Ruler</td>
<td>24</td>
</tr>
<tr>
<td>2.1.9.3 Labeled Electrodes</td>
<td>25</td>
</tr>
<tr>
<td>2.1.9.4 MEMS Comb Drive Flexures</td>
<td>26</td>
</tr>
<tr>
<td>2.1.9.5 MEMS Radial Comb Drive Circular Hub</td>
<td>27</td>
</tr>
<tr>
<td>2.1.9.6 Scripting Example Files Description</td>
<td>28</td>
</tr>
</tbody>
</table>

2.2 Interface Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.1 Comments</td>
<td>32</td>
</tr>
<tr>
<td>2.2.2 Creating Structures (Cells)</td>
<td>32</td>
</tr>
<tr>
<td>2.2.3 Layer</td>
<td>32</td>
</tr>
<tr>
<td>2.2.4 Data Type</td>
<td>32</td>
</tr>
<tr>
<td>2.2.5 GDS Rendering Resolution</td>
<td>32</td>
</tr>
<tr>
<td>2.2.6 Shape Resolution</td>
<td>33</td>
</tr>
</tbody>
</table>
Chapter 1 CONTENTS

2.7.3 Fractals ........................................... 74
2.7.4 Function Plot ........................................... 75
2.7.5 Grayscale ........................................... 77
   2.7.5.1 Polygons ........................................... 77
   2.7.5.2 Ramp ........................................... 79
   2.7.5.3 Spiral Staircase ........................................... 81
2.7.6 Interdigitated Electrodes ........................................... 83
2.7.7 Junctions ........................................... 86
   2.7.7.1 T Junction ........................................... 86
   2.7.7.2 H Junction ........................................... 87
   2.7.7.3 Arrow Junction ........................................... 88
2.7.8 Meander Channel ........................................... 89
2.7.9 Points To Shape ........................................... 92
2.7.10 Polygon Along a Path ........................................... 93
2.7.11 Random Polygons ........................................... 94
2.7.12 Random Ellipses and Vectorized Ellipses ........................................... 95
2.7.13 Resolution Test Pattern ........................................... 96
2.7.14 Spiral ........................................... 99
2.7.15 Spiral - Rectangular ........................................... 100

2.8 Alignment and Reticle Elements ........................................... 101
   2.8.1 Alignment Marks - Predefined ........................................... 101
   2.8.2 Alignment Marks - Custom ........................................... 105
   2.8.3 Reticle Barcode and Label Frames ........................................... 107
   2.8.4 Vernier ........................................... 108
   2.8.5 Vernier With Labels ........................................... 109
   2.8.6 Arrows ........................................... 110

2.9 CNST Nanophotonics Library ........................................... 111
   2.9.1 Waveguides ........................................... 112
      2.9.1.1 Waveguide ........................................... 112
      2.9.1.2 Waveguide Slot ........................................... 113
      2.9.1.3 Waveguide Inverse ........................................... 114
      2.9.1.4 Waveguide Inverse Slot ........................................... 115
      2.9.1.5 Waveguide Expander ........................................... 116
   2.9.2 Tapers ........................................... 117
      2.9.2.1 Linear Taper ........................................... 117
      2.9.2.2 Linear Taper Slot ........................................... 118
      2.9.2.3 Linear Taper Inverse Slot ........................................... 119
      2.9.2.4 Exponential Taper ........................................... 120
      2.9.2.5 Exponential Taper Inverse ........................................... 121
      2.9.2.6 Exponential Taper Inverse Slot ........................................... 122
      2.9.2.7 Custom Taper ........................................... 123
   2.9.3 Couplers ........................................... 124
      2.9.3.1 Directional Couplers ........................................... 124
      2.9.3.2 S-Bend Taper ........................................... 128
      2.9.3.3 S-Bend Funnel ........................................... 129
      2.9.3.4 S-Bend ........................................... 130
Chapter 1  CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9.3.5</td>
<td>S-Bend Inverse</td>
<td>131</td>
</tr>
<tr>
<td>2.9.3.6</td>
<td>S-Bend Inverse Slot</td>
<td>132</td>
</tr>
<tr>
<td>2.9.3.7</td>
<td>Y-Bend</td>
<td>133</td>
</tr>
<tr>
<td>2.9.3.8</td>
<td>Y-Bend Inverse</td>
<td>134</td>
</tr>
<tr>
<td>2.9.3.9</td>
<td>Y-Bend Inverse Slot</td>
<td>135</td>
</tr>
<tr>
<td>2.9.3.10</td>
<td>Y-Bend - 90 degree</td>
<td>136</td>
</tr>
<tr>
<td>2.9.3.11</td>
<td>Y-Bend Inverse - 90 degree</td>
<td>137</td>
</tr>
<tr>
<td>2.9.3.12</td>
<td>Y-Bend Inverse Slot - 90 degree</td>
<td>138</td>
</tr>
<tr>
<td>2.9.3.13</td>
<td>90 Degree Bend</td>
<td>139</td>
</tr>
<tr>
<td>2.9.3.14</td>
<td>90 Degree Bend Inverse</td>
<td>140</td>
</tr>
<tr>
<td>2.9.3.15</td>
<td>90 Degree Bend Inverse Slot</td>
<td>141</td>
</tr>
<tr>
<td>2.9.3.16</td>
<td>180 Degree Bend</td>
<td>142</td>
</tr>
<tr>
<td>2.9.3.17</td>
<td>180 Degree Bend Inverse</td>
<td>143</td>
</tr>
<tr>
<td>2.9.3.18</td>
<td>180 Degree Bend Inverse Slot</td>
<td>144</td>
</tr>
<tr>
<td>2.9.4</td>
<td>Racetrack</td>
<td>145</td>
</tr>
<tr>
<td>2.9.5</td>
<td>Spiral Delay Line</td>
<td>146</td>
</tr>
<tr>
<td>2.9.6</td>
<td>Inverse Spiral Delay Line</td>
<td>148</td>
</tr>
<tr>
<td>2.9.7</td>
<td>Gratings</td>
<td>150</td>
</tr>
<tr>
<td>2.9.7.1</td>
<td>Grating</td>
<td>150</td>
</tr>
<tr>
<td>2.9.7.2</td>
<td>Apodized Grating</td>
<td>151</td>
</tr>
<tr>
<td>2.9.7.3</td>
<td>Grating Coupler</td>
<td>152</td>
</tr>
<tr>
<td>2.9.7.4</td>
<td>Grating Couplers With Waveguides</td>
<td>153</td>
</tr>
<tr>
<td>2.9.8</td>
<td>Photonic Crystals and Hexagonal Arrays</td>
<td>156</td>
</tr>
<tr>
<td>2.9.9</td>
<td>Disc-Ring Architectures</td>
<td>157</td>
</tr>
<tr>
<td>2.9.9.1</td>
<td>Disc-Ring - Bezier Curves, Arcs and Endcaps</td>
<td>157</td>
</tr>
<tr>
<td>2.9.9.2</td>
<td>Disc-Ring Infinite</td>
<td>159</td>
</tr>
<tr>
<td>2.9.9.3</td>
<td>Disc-Ring Infinite Inverse</td>
<td>161</td>
</tr>
<tr>
<td>2.9.9.4</td>
<td>Disc-Ring Infinite Inverse Positive Tone</td>
<td>163</td>
</tr>
<tr>
<td>2.9.9.5</td>
<td>Disc-Ring Symmetric Bezier</td>
<td>165</td>
</tr>
<tr>
<td>2.9.9.6</td>
<td>Disc-Ring Symmetric Arc</td>
<td>175</td>
</tr>
<tr>
<td>2.9.9.7</td>
<td>Disc-Ring Symmetric Inverse Bezier</td>
<td>185</td>
</tr>
<tr>
<td>2.9.9.8</td>
<td>Disc-Ring Symmetric Inverse Arc</td>
<td>195</td>
</tr>
<tr>
<td>2.9.9.9</td>
<td>Disc-Ring Symmetric Inverse Positive Tone Bezier</td>
<td>205</td>
</tr>
<tr>
<td>2.9.9.10</td>
<td>Disc-Ring Symmetric Inverse Positive Tone Arc</td>
<td>215</td>
</tr>
<tr>
<td>2.9.9.11</td>
<td>Disc-Ring Pulley Bezier</td>
<td>225</td>
</tr>
<tr>
<td>2.9.9.12</td>
<td>Disc-Ring Pulley Arc</td>
<td>235</td>
</tr>
<tr>
<td>2.9.9.13</td>
<td>Disc-Ring Pulley Inverse Bezier</td>
<td>245</td>
</tr>
<tr>
<td>2.9.9.14</td>
<td>Disc-Ring Pulley Inverse Arc</td>
<td>255</td>
</tr>
<tr>
<td>2.9.9.15</td>
<td>Disc-Ring Pulley Inverse Positive Tone Bezier</td>
<td>265</td>
</tr>
<tr>
<td>2.9.9.16</td>
<td>Disc-Ring Pulley Inverse Positive Tone Arc</td>
<td>275</td>
</tr>
<tr>
<td>2.9.10</td>
<td>Waveguide Inverse Photonic Crystals (Ellipse)</td>
<td>285</td>
</tr>
<tr>
<td>2.9.11</td>
<td>Waveguide Photonic Crystals (Ellipse)</td>
<td>286</td>
</tr>
<tr>
<td>2.9.12</td>
<td>Waveguide Inverse Photonic Crystals (Rectangle)</td>
<td>287</td>
</tr>
<tr>
<td>2.9.13</td>
<td>Waveguide Photonic Crystals (Rectangle)</td>
<td>288</td>
</tr>
<tr>
<td>2.9.14</td>
<td>Waveguide Photonic Crystals (Flush Rectangle)</td>
<td>289</td>
</tr>
<tr>
<td>2.9.15</td>
<td>Various Waveguide-Disc-Tip Coupled Structures</td>
<td>290</td>
</tr>
</tbody>
</table>
Chapter 1 CONTENTS

2.10 MEMS - NEMS Library ........................................ 298
  2.10.1 Actuators ................................................. 298
    2.10.1.1 Bent Beams ....................................... 298
    2.10.1.2 Bi-Morph Thermal Actuator ....................... 299
    2.10.1.3 Combs and Drive Elements ....................... 300
    2.10.1.4 Folded Springs ................................. 302
  2.10.2 Bolometers .............................................. 311
  2.10.3 Gears .................................................... 313
    2.10.3.1 Hub With Straight and Circular Springs ....... 314
    2.10.3.2 Radial Comb Drive .............................. 317
  2.10.4 Anchored Flexures ...................................... 319
  2.10.5 Cantilevers ............................................. 329
  2.10.6 Doubly Clamped Beams .................................. 340
  2.10.7 Interacting Arrays ..................................... 347
  2.10.8 Stress, Strain Measurement Structures ............... 357
    2.10.8.1 Guckel Rings .................................. 357
    2.10.8.2 Diamond Ring ................................... 358
  2.10.9 Suspended Fluid Cell ................................... 359

Graphical User Interface ............................................. 360
  3.1 Basic Shapes ............................................... 360
    3.1.1 Pillar-Hole (Square/Hex) Array ..................... 360
    3.1.2 Torus ............................................... 360
    3.1.3 Grating Coupler - Bulls Eye ....................... 360
    3.1.4 Spirals .............................................. 361
    3.1.5 Gratings ............................................ 361
  3.2 Lithography Machine Resources .............................. 363
    3.2.1 CNST Reticle Frame Generator ....................... 363
    3.2.2 Generic Reticle Frames .............................. 363
    3.2.3 Ebeam Lithography - Job and Schedule File Generator 363
      3.2.3.1 Default Values Initialization File .......... 363
      3.2.3.2 Main ........................................... 363
      3.2.3.3 Align .......................................... 364
      3.2.3.4 Pattern ........................................ 364
      3.2.3.5 Dose Matrix ................................... 365
    3.2.4 EBL Alignment Offset .................................. 368
    3.2.5 EBL Max Clock ....................................... 368
    3.2.6 EBL Write Time Estimation ............................ 368
  3.3 Advanced CAD Resources ..................................... 370
    3.3.1 Label Maker .......................................... 370
    3.3.2 Text To GDS ......................................... 370
    3.3.3 Arbitrary Function Generator ...................... 370
    3.3.4 Binary Zone Plate ................................... 371
    3.3.5 Photonic Crystals .................................. 372
    3.3.6 Random Polygons .................................... 372
    3.3.7 Random Rectangular Array ........................... 373
3.3.8 Cantilever Arrays ........................................... 373
3.3.9 Verniers ..................................................... 374
3.3.10 Fractals .................................................... 374
3.3.11 Grayscale Image To GDS ................................. 374

Programming Reference ........................................ 376
 4.1 Programming Examples ....................................... 376
  4.1.1 Template .................................................. 376
  4.1.2 Script Method Access .................................... 377
  4.1.3 Boolean Operations and Affine Transformations .... 379
  4.1.4 Labeled Arrays .......................................... 381
  4.1.5 MEMS Perforated Flexures .............................. 383
  4.1.6 Custom Class Methods .................................. 385
  4.1.7 GDS Area Objects - Layers and Data-types ........ 387
  4.1.8 Centering GDS Area Objects ............................ 388
  4.1.9 PostScript to GDS ........................................ 389
  4.1.10 Curved Fluidic Channels ............................... 391

 4.2 Reference Methods .......................................... 394
  4.2.1 Accessor and Mutator Methods ......................... 394
  4.2.2 Miscellaneous GDS Area and Struct Methods ....... 395
  4.2.3 Shape Methods .......................................... 396
  4.2.4 Array Methods .......................................... 398
  4.2.5 Text, Labels, PostScript and Logo Methods ........ 401
  4.2.6 Miscellaneous Object Methods ......................... 404
  4.2.7 Alignment and Reticle Element Methods ............. 414
  4.2.8 Nanophotonics Library Methods ....................... 417
  4.2.9 MEMS NEMS Library Methods ......................... 453

Bibliography ..................................................... 479
1.1 Overview

Platform independent CNST Nanolithography toolbox for scripted layout generation and complex processing was designed to address aggressively scaled nanoscale device architectures. Within the layout design phase, imprecise representation of curved objects [1–37] at nanoscale dimensions results in increased line-edge roughness. Scattering events from the consequent asperities along device peripheries leads to enhanced dissipative effects. To circumvent these dilemmas, along with standard commercial layout tools, we have developed a custom CAD scripting software package for directly streaming complex shapes to GDSII.

At CNST, we have strong research efforts in nanoscale optics and photonics [38–69]. Consequently, we have developed a library of nanophotonics components (the CNST nanophotonics library contains microrings, S-bends based on Bezier curves, photonic crystals, variety of tapers, grating couplers [70–75] and other complex structures) with precise control of vertex location in the GDSII file. Accurate shape representation is of particular importance to high-resolution lithography, for example, using the either of the two NanoFab 100keV tools with sub-nm grid snapping capability. The flexible scripting interface enables significant user customization to lay out structures outside of the standard available designs. Furthermore, the toolbox is platform independent and will run on any operating system (Linux, Windows and MacOS X) with Java standard edition (SE) 8. In addition to standard Java libraries, the toolbox utilizes the freely available JGDS library for encoding shapes to GDS objects [76].

The CNST nanolithography toolbox also contains primitives (ellipses, torus, rectangles, etc), pillar-hole arrays (rectangular, hexagonal), a variety of spirals, fractals, gratings, verniers, arbitrary function generator (shapes defined by a mathematical function), label makers, text to GDS, postScript to GDS, zone plates, grayscale objects [77–82], images to GDS, polar arrays (also, hexagonal and rectangular), and many more customized shapes. Additionally, we include a MEMS library of elements containing variety of actuators, flexures, clamped beams, and interacting array structures [83–103]. The package also contains CNST-developed reticle generators for various steppers (barcode, label, reticle marks), a CNST ebeam lithography write time estimator, and a ebeam lithography job and schedule file generator with a graphical display of pattern placement (Figure 1.1).
Figure 1.1: CNST Nanolithography Toolbox chart illustrating a myriad of available complex shapes, objects and functions.
1.1.1 Terms Of Use

This software was developed at the National Institute of Standards and Technology (NIST) by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain. The NIST CNST nanolithography toolbox is an experimental system. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. We would appreciate acknowledgment if the software is used. This software can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified.

1.1.2 Disclaimer

This manual identifies certain commercial equipment, instruments, and materials to specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment, instruments, and materials identified are necessarily the best available for the purpose.

1.1.3 Acknowledgements

If you developed layouts using the CNST nanolithography toolbox, please acknowledge its use by including the following reference:

1.1.4 Development Team

Many researchers at NIST, CNST and external collaborators have made substantial contributions to the nanolithography toolbox. The development team most notably consists of Krishna C. Balram (University of Maryland), Daron A. Westly, Marcelo Davanco, Karen Grutter, Qing Li (University of Maryland), Thomas Michels, Christopher H. Ray, Liya Yu, Richard Kasica, Christopher B. Wallin (University of Maryland), Ian Gilbert (University of Maryland), Brian A. Bryce (Harvey Mudd College), Gregory Simelgor (Edico Genome), Juraj Topolancik (Roche Sequencing Solutions), Nicolae Lobontiu (University of Alaska), Yuxiang Liu (Worcester Polytechnic Institute), Pavel Neuzil (Brno University of Technol-
ogy, KIST-Europe), Vojta Svatos (Brno University of Technology), Kristen A. Dill, Neal A. Bertrand, Meredith Metzler (University of Pennsylvania), Gerald Lopez (University of Pennsylvania), David A. Czaplewski (Argonne National Laboratory), Leonidas Ocola (Argonne National Laboratory), Kartik Srinivasan, Samuel Stavis, Vladimir Aksyuk, J. Alexander Liddle, Slava Krylov (Tel-Aviv University) and B. Robert Ilic.

1.1.5 Future Developments

The CNST nanolithography toolbox is broad in scope and continuously growing within the CNST community. The toolbox is distributed with the hope that it will be useful, but without any warranty, without even an implied warranty for any particular purpose. All efforts have been made to ensure that the CNST nanolithography toolbox is supported on Linux, Windows and MacOS X. Please direct comments, suggestions, encountered bugs to Nanolithography.Toolbox@nist.gov. The software will evolve over time, implementing features as the CNST identifies and prioritizes new applications.
1.1.6 Java 8 Requirements

The graphical user interface (GUI) was constructed using JavaFX components. The code employs a variety of elements contained in latest java standard edition (SE) 8, hence earlier versions will not work correctly. To determine the SE version, at the (windows command or linux/mac terminal) prompt type:

    java --version

At the terminal prompt the following would appear:

    java version "1.8.0_40"
    Java(TM) SE Runtime Environment (build 1.8.0_40-b26)
    Java HotSpot(TM) 64-Bit Server VM (build 25.40-b25, mixed mode)

1.8.X_XX indicates Java version 8 is available. To run the CNST NanoLithography Toolbox, either double click on the CNSTnanoToolboxVXXXX.jar file (X's represent version numbers) or run from a command prompt in a following manner:

    java --jar CNSTnanoToolboxVXXXX.jar

Upon execution of the java code, the CNST Nanolithography Toolbox will appear in a similar fashion as seen in figure 1.2. The menu appearance may vary slightly depending on the operating system the interface is running under. Figure 1.2 shows the toolbox as it appears under 64-bit Windows 7.

1.1.7 Distribution Package

The package contains the following files and directories:

- CNSTnanoToolboxVXXXX.jar  The CNST Nanolithography Toolbox executable Java JAR file. X's represent version numbers.
- CNSTdefaultValues.xml  Default parameter values for the CNST ebeam tool module, load and save file directories.
- Examples  Directory containing syntax coloring XML language definition file, numerous scripting, GUI module and programming examples.

1.1.8 Modes of Operation

1.1.8.1 CNST Scripting

The most powerful feature of the CNST nanolithography toolbox is scripting. Virtually any shape can be constructed using the CNST Scripting features. The scripting feature is listed as the first option within the resource menu. The scripting module reads in an ASCII file with a .cnst extension and converts
complex objects directly to GDS. We also provide an extensive library of objects including numerous nanophotonic elements. During the pattern design phase, space is broken up into grid points between which the CNST scripting objects are rendered. Figure 1.3 shows various points connecting a variety of generated nanophotonic elements (exponential taper, photonic crystal waveguide, y-splitters, waveguides, s-bends and rings). Syntax coloring for .cnst files is available using NotePad++. This feature improves readability-structure and errors become visually distinct.

CNST scripting offers a myriad of options that are either not found or extremely difficult to implement within standard CAD packages that produce GDS files. The full command reference for CNST scripting is included in section 1.1.8.2. Figure 1.4 illustrates a variety of generated GDS shapes. CNST scripting is either carried out from the GUI or from the terminal command prompt (see section 2.1). Additionally, running batch execution of many .cnst files is described in section 2.1.5.
Figure 1.3: Pattern layout design schematic highlighting various available nanophotonic elements.

Figure 1.4: CNST Scripting example illustrating a variety of available shapes streamed directly to GDSII.
1.1.8.2 Graphical User Interface

The main window of the graphical used interface is divided into three parts. The top title panel has the CNST logo and is static. The left panel contains a collapsible tree with options representing available toolbox features. The large right panel is the dynamic canvas displaying selected module options.

Double clicking on a module feature or single clicking on the triangle directly to the left that node expands the menu options. Consequently, all module elements within that branch node are displayed. For instance, the Basic Shapes branch has several objects including polygon-based pillar-hole arrays, arrays of tori, grating couplers, spirals and gratings. Highlighting any of the accessible feature nodes will display available options of the specified module. In case pillar-hole arrays, user defined parameters within a text field such as GDS file name, number of sides (shape vertices), GDS layer, rotation, radius, object array options (position, number of elements and pitch) along with buttons to generate a GDS file with the specified options, an about button, and an Exit button. Figure 1.5 shows a highlighted Pillar-Hole (Square/Hex) Array branch node within the left panel. The main panel shows various available module parameters within the toolbox GUI.

![Graphical User Interface](image)

**Figure 1.5:** Pillar-hole array module example showing various available options within the main panel of the toolbox GUI.
2.1 Setup, Execution and Examples

Scripting execution takes place either from the graphical user interface or the terminal command prompt.

2.1.1 Quick Start Setup

1. Change the <OpenDirectory> and <SaveToDirectory> parameters in the CNSTdefaultValues.xml setup file (see section 2.1.2).

2. Setup syntax coloring within NotePad++ (see section 2.1.8).

3. Create script files with extension .cnst. See scripting examples in section 2.1.9.6 and constructors in this chapter.

4. Run scripts either within the graphical user interface (section 2.1.3) or via command prompt (section 2.1.4).
2.1.2 Setup - CNSTdefaultValues.xml open and save directories

The CNST NanoLithography ToolBox requires that the CNSTdefaultValues.xml setup file resides in the same directory as the Java executable .jar file.

The first step in running and executing scripts requires the modification of the open and save directory variables within the CNSTdefaultValues.xml file. The toolbox will not function on some operating systems if these values are not properly changed.

Open the CNSTdefaultValues.xml file within a text editor and change the <OpenDirectory> and <SaveToDirectory> parameters to a directory where .cnst script and GDS files are respectively read and saved. Figures 2.6 and 2.6 illustrate modifications for Windows and Mac/Unix/Linux platforms.

(a)

<OpenDirectory>C:\Users\user\CNSTnanoToolBox\loadFiles\</OpenDirectory>
<SaveToDirectory>C:\Users\user\CNSTnanoToolBox\saveFiles\</SaveToDirectory>

(b)

<OpenDirectory>/Users/user/CNSTnanoToolBox/loadFiles/</OpenDirectory>
<SaveToDirectory>/Users/user/CNSTnanoToolBox/saveFiles/</SaveToDirectory>

Figure 2.6: Modification to open and save directory variables within the CNSTdefaultValues.xml setup file for (a) Windows and (b) Mac/Unix/Linux platforms.
2.1.3 Graphical User Interface Execution

Start the graphical user interface by either double clicking on the executable jar file or at the prompt type:

```
java -jar CNSTnanoToolboxVXXXX.jar
```

When the interface appears, in the left panel click CNST Scripting (step 1) and then click the Load Script button (step 2). Within the file manager locate the .cnst script file (step 3), click Open (step 4) and the full path of the script file will appear in the Script File: label. Type in the desired GDS file name and click the Create GDS button. Under the STATUS: label, the full path of the saved GDS file will appear.

Figure 2.7: GDS generation using the graphical scripting interface.
2.1.4 Running CNST Scripts From a Terminal Command Prompt

At the terminal command prompt, the command below allows script execution without the graphical user interface. Additionally, using the `getfonts` option allows the extraction of all available system font names to be stored into atime/date stamped ASCII file.

```
java -jar CNSTnanoToolboxVXXXX.jar [option]
```

[option]
- `cnstscripting inputFileName.cnst outputFileName.gds`
  Process a CNST scripting file `inputFileName.cnst` and cast the resulting structures into an `outputFileName.gds` file.

- `getfonts`
  Obtain system fonts and store them into a time/date stamped file. Useful for correctly spelling font names within various available text constructors.

2.1.5 Running Multiple CNST Scripts - Batch File Mode

Batch mode execution of many CNST script files is performed from a terminal command prompt. In Windows, to open the command prompt, click Start then type cmd within the ‘search programs and files’. Within the Linux and Mac environments open a terminal command prompt. First create a text file where each line entry has the format defined within the previous section (2.1.4). In Windows, name the file with an extension BAT, i.e. `CNSTbatchFile.BAT`. For instance, the text file would have the following entries:

```
java -jar CNSTnanoToolboxVXXXX.jar cnstscripting scriptFile01.cnst output01.gds
java -jar CNSTnanoToolboxVXXXX.jar cnstscripting scriptFile02.cnst output02.gds
...```

To execute the ASCII batch file, at the windows DOS prompt type the file name:

```
C:\Users\robilic\CNSTscripting>CNSTbatchFile.BAT
```

In Linux or Mac type dot then backslash followed by the filename:

```
robilic\CNSTscripting $>.\CNSTbatchFile.BAT
```

NIST • CNST Nanolithography Toolbox v2016.09.01 • http://www.nist.gov/cnst/ page 19 of 488
2.1.6 Running CNST Scripts Within Matlab

Matlab allows for execution of system commands and return outputs. Figure 2.1.6 shows how matlab generates a `matlabScript.cnst` file. Subsequently, using a system command, matlab runs `java` to create a GDS file. This was accomplished by using commands described in section 2.1.4. Lastly, another system command executes kLayout, an open source layout viewer and editor, in order to display the output GDS file. The generated file has two GDS structures. The first has a single ellipse, and the second has 100 ellipses, each cast into a different GDS layer, along a diagonal. The latter was generated using the `for` loop control statement.

The return status shows the command prompt output during code execution. In this case, the directory where the output GDS resides (see save directory information in section 2.1.2).

```matlab
fileID = fopen('matlabScript.cnst','w');
fprintf(fileID,'0.001 gdsReso
');
fprintf(fileID,'0.001 shapeReso

');
fprintf(fileID,'# Creating a simple ellipse with Matlab
');
fprintf(fileID,'<matlabEllipse struct>
');
fprintf(fileID,'11 layer
');
fprintf(fileID,'0 0 2 4 44 0 ellipse

');
% array of 100 ellipses along a diagonal with different GDS layers
fprintf(fileID,'# Creating array of 100 ellipses along the diagonal
');
fprintf(fileID,'<matlabEllipseArray struct>
');
for i=1:100
    fprintf(fileID,'%s%s
' , int2str(i), ' layer');
    fprintf(fileID,'%s%s%s%s
', int2str(i), ' ', int2str(i), ' 0.4 44 0 ellipse');
end
fclose(fileID);
% run the java code from matlab
[status,cmdout] = dos('java -jar CNSTnanoToolbox.jar cnstscripting matlabScript.cnst matlabGDSoutput.gds');
status, cmdout
% open GDS with klayout - use quotes around directory names with spaces
[status,cmdout] = dos('"C:\Program Files\KLayout (64bit)\klayout* D:\\DOCUMENTS\zTESTfiles\saveFiles\matlabGDSoutput.gds"');
status, cmdout
```

Figure 2.8: GDS generation and viewing using a Matlab script.

Example matlab files are located in `\EXAMPLES\CNSTscripting\Matlab`. 
2.1.7 Python Script Generation

Python programming can be used to create complex CNST scripting files. Programming in general allows for variable initialization, loops, Booleans, branching, control statements, conditionals and other methods to be used when algorithmically constructing script files. The program shown in figure 2.9 creates a pythonScript.cnst script file that contains two GDS structures, one with a single ellipse, and another with an array of circles along a circular path in another GDS structure. The latter is accomplished within a for loop repetition statement. Subsequently the nanolithography toolbox is used to create an output GDS file seen as an inset in figure 2.9. Example python files are located in \EXAMPLES\CNSTscripting\Python.

```python
# -*- coding: utf-8 -*-
""
@author: rob
""
import math
f = open('pythonScript.cnst', 'w')
f.write(str('0.001 gdsReso
'))
f.write(str('0.001 shapeReso

'))
f.write(str('# Creating a simple ellipse with Python'))

f.write(str('<pythonEllipse struct>
'))
f.write(str('11 layer
'))
f.write(str('0 0 2 4 44 0 ellipse
'))

f.write(str('# 100 circles along a circular path
'))
for x in xrange(num):
    xCoord = radius * math.cos(x*increment)
    yCoord = radius * math.sin(x*increment)
    f.write(str('%.4f %.4f 0.2 0.2 44 0 ellipse
' % (xCoord, yCoord)))

f.close

Figure 2.9: Python programming example used to generate an array of 100 circles, each within a distinct GDS layer, along a circular path. Inset shows the rendered GDS output of circles along a circular path.
2.1.8 NotePad++ Syntax Coloring

The package provides a language definition XML file for Notepad++. The constructor keywords listed within the XML file could be used to create a language definition file for any text editor. Syntax coloring within NotePad++ is accomplished by reading in the provided CNSTscripting.xml language definition file. Subsequently, all files with extension .cnst will have proper syntax coloring (Figure 2.10). This is useful when visualizing and debugging scripts. The following is a procedure for NotePad++ syntax coloring:

1. Language -> Define your language.... -> click Import....
2. choose \EXAMPLES\CNSTscripting\NotePad++XML\CNSTscripting.xml language definition file and click open
3. close NotePad++ and reopen a file with extension .cnst
4. Some versions may require users to repeat this procedure

Figure 2.10: Notepad++ screenshot illustrating syntax coloring of .cnst script files defined by the CNSTscripting.xml language definition file.
2.1.9 Scripting Examples

2.1.9.1 Basic Scripting Example

Figure 2.11 shows a simple script that creates two structures, one with a rectangle and the other with two ellipses. The procedure for creating script files is the following:

1. `gdsReso` defines the output rendering resolution (in µm) of the GDS file. This parameter is included at the top of each file.

2. `shapeReso` defines the rendering resolution (in µm) of vectorized shapes. This parameter can be placed anywhere in the file and can be changed for each vectorized shape.

3. Create structures.

4. Initialize layers and dataTypes, then place objects into structures. The following subsections introduce a variety of constructors for creating complex shapes.

```plaintext
0.001 gdsReso
0.001 shapeReso

# create a GDS structure/cell to store shapes
<simpleRectangles struct>

# set GDS layer
4 layer

# create a rectangle
-5 20 0 28 0 rectangle

# create another structure
<circlesAndEllipses struct>

# create an ellipse
0 0 40 100 88 22.0 ellipse

# create ellipse in GDS layer 40 with a datatype 44
40 layer
44 dataType
100 80 40 40 44 0 ellipse
```

Figure 2.11: Basic scripting example (`scriptingExampleBasicScriptingExample.cnst`).
2.1.9.2 Scripting Example - Labeled Calibration Ruler

The example script creates a labeled calibration ruler. Tick marks are 250 nm wide and 5 \( \mu m \) long. Markers labeled 5 and 10 markers are 7.5 \( \mu m \) and 10 \( \mu m \) long. This type structure is useful for calibrating (optical and electron) microscopes.

```
0.001 gdsReso
0.001 shapeReso
# layer
## Example - Labelled Calibration Ruler

##### tick marks 5um long and 250nm wide
<tickMark struct>
-0.125 0 0.125 5 0 rectangle
##### tick mark extension 2.5um long and 250nm wide
<tickMarkExtension struct>
-0.125 0 0.125 2.5 0 rectangle
<calibrationRuler struct>
# 20um long
<tickMark 0 0 21 1 1 0 1 arrayRect>
# add tick mark extensions every 5um
<tickMarkExtension 5 5 4 1 5 0 1 arrayRect>
# add tick mark extensions every 10um
<tickMarkExtension 10 7.5 2 1 10 0 1 arrayRect>

# tick mark label (TAB delimited)
{0 5 10 15 20um 15Arial 2 -0.5 -2 0051 rowCol labelMaker}
```

(a)

(b)

Figure 2.12: Labeled calibration ruler example. (a) Example script (`scriptingExampleLabeledCalibrationRuler.cnst`) illustrates the construction of (b) a labeled calibration ruler.
2.1.9.3 Labeled Electrodes

The script initiates with a 250µm bond pad. The structure is then instantiated into a 5 × 1 rectangular array, Bezier curves and alignment crosses are then added. This structure (electrodeSegment) is then instantiated to construct the electrodeHalf, which is then used to form the electrodes structure. Bond pad labels are then added and electrodes structure is instantiated into top using the instanceSym constructor, ensuring symmetric extents around the origin.

```
0.001 gdsReso
0.01 shapeReso

# 250µm bondpad
<bondPad250um struct>
0 0 250 250 rectangle

# 1/8 electrode
<electrodeSegment struct>
<bondPad250um 0 0 5 1 500 1 1 arrayRect>
625 250 625 900 625 2200 625 2200 2150 20 0 bezierCurve
<bondPad250um 1125 250 1125 900 2250 625 2250 2150 20 0 bezierCurve
<bondPad250um 1625 250 1625 900 2300 625 2300 2150 20 0 bezierCurve
<bondPad250um 2125 250 2125 900 2350 625 2350 2150 20 0 bezierCurve

# JEOL alignment crosses
500 100 10 60 cross
750 100 10 60 cross
1000 100 10 60 cross
1250 150 10 60 cross
1500 150 10 60 cross

# 1/2 electrode
<electrodeHalf struct>
<electrodeSegment 0 0 N 1 0 instance>
<electrodeSegment 4750 0 Y 1 0 instance>
<electrodeSegment 0 0 Y 1 -90 instance>
<electrodeSegment 4750 0 Y 1 90 instance>

# electrode set
<electrodes struct>
<electrodeHalf 0 0 N 1 0 instance>
<electrodeHalf 0 4750 X 1 0 instance>

# electrode labels top and bottom - Letters
{0 8 Serif 100 410 75 0 0 500 0 autoOutLett labelMaker}
{0 8 Serif 100 410 4275 0 0 500 0 autoOutLett labelMaker}

# electrode labels left and right - Numbers
{8 0 Serif 100 0 0 100 4275 0 500 autoOutLett labelMaker}
{8 0 Serif 100 0 0 4600 4275 0 500 autoOutLett labelMaker}

# top cell centered electrode cell using instanceSym
<top struct>
<electrodes 0 0 0 4750 4750 N 1 0 instanceSym>
```

Figure 2.13: Scripting example (scriptingExampleElectrode.cnst) illustrates the construction of electrodes with labels and 5 levels of alignment crosses.
2.1.9.4 MEMS Comb Drive Flexures

The example script below first creates a microelectromechanical systems (MEMS) flexure with two anchored pads using the `flexure2E` constructor. Subsequently, two linear comb drive structures are connected to the flexure element.

```
0.001 gdsReso
0.001 shapeReso
\4 layer

# Example - MEMS Flexure With Comb Drive Elements

<memsExample struct>
  # Flexure
  0 0 8.2 0.2 0.88 0.22 2.44 20 8 1.44 2.42 3.4 0.2 8.44 4.4 0.4 7 0 flexure2E
  # Top Comb
  -5.15 29.48 0.4 3.4 10.1 9.1 10 1.1 4.1 31 0.2 7 0 combDriveV1
  # Bottom Comb
  5.15 -29.48 0.4 3.4 10.1 9.1 10 1.1 4.1 31 0.2 7 180 combDriveV1
```

Figure 2.14: MEMS linear comb drive actuating flexure example. (a) Example script (`scriptingExampleMEMSv1.cnst`) illustrates the construction of (b) comb drive elements connected to a movable flexure.
2.1.9.5 MEMS Radial Comb Drive Circular Hub

The example script below first creates four radial comb structures connected to a circular hub.

```
0.001 gdsReso
0.001 shapeReso

# Example - MEMS Flexure With Comb Drive Elements

<circlespringExample struct>
# circular spring
  0 0 1 4 3.4 40 128 18 0.2 7 0 circlespring
# radial combs
  10 0 4 40 1 30 1.1 2.2 10 32 40 20 1 7 0 combRadialV2
  0 10 4 40 1 30 1.1 2.2 10 32 40 20 1 7 90 combRadialV2
  -10 0 4 40 1 30 1.1 2.2 10 32 40 20 1 7 180 combRadialV2
  0 -10 4 40 1 30 1.1 2.2 10 32 40 20 1 7 270 combRadialV2
```

Figure 2.15: MEMS radial comb drive actuator example. (a) Example script (scriptingExampleMEMSv2.cnst) used to create (b) four radial comb drive elements connected to a circular hub.
2.1.9.6 Scripting Example Files Description

CNST scripting offers a variety of constructor objects that allow for any conceivable shape to be rendered into a GDS output. This chapter includes a number of CNST scripting examples designed to elaborate on the functionality of these objects. Overall, the example collection covers each CNST scripting constructor. To improve the readability of the script, we included brief constructor documentation comments in each of the presented examples. The example scripts (.cnst files) and the corresponding GDS output files are contained in a folder \loadFiles\CNST scripting. The folder also contains an Excel spreadsheet CNST scripting.xlsx with data used in several presented examples. A brief description of each example script file is given below.

scriptingAlignmentFeatures.cnst contains examples of variety of multi-level preconfigured and custom user defined alignment mark strategies, verniers, and arrows. Details of these features are described in section 2.8.

scriptingArraysAndInstancing.cnst creates a structure with the name myPattern constructed from various primitive elements, including text and spiral objects. The structure is then instantiated at a 10X reduced magnification. Furthermore, the structure is arranged into rectangular, polar and hexagonal arrays.

scriptingBasicShapes.cnst is a simple script demonstrating the use of primitive shapes. First, we introduce line comments. Then, resolution features of the GDS rendered output and vectorized shapes is explained. Next, we present constructors for creating GDS structures (cells) and layers. Directly following, ellipses, vectorized ellipses, tori (arcs), rectangles, rounded rectangles, polygons, star shapes, and crosses are constructed within initialized GDS structures and layers.

scriptingBezierCurve.cnst illustrates the use of the Bezier curve constructor.

scriptingBooleansGenAreasBiasTransformations.cnst is an example demonstrating extraction of layer shapes from a GDS structure and casting the resulting shapes into a generalized area object. Using these objects boolean operations, general area copies, shape biasing and affine transformation of the areas is demonstrated.

scriptingCNSTasm1ContactLabelBarcodeGenerator.cnst demonstrates the use of various constructors used to generate contact lithography (label and logo) and CNST ASML PAS5500 i-line stepper reticles (reticle marks, label, barcode and logo).

scriptingCircleTorusWave.cnst example demonstrates circular (section 2.3.3) and torus (section 2.3.16) structures with sinusoidally varying boundaries.
scriptingCustomTaper.cnst example employs data from the CNSTscripting.xlsx file (CustomTaper sheet) to create a taper defined by the \((x,y)\) point pairs. Data was simply copied and pasted into the cns file, then terminated by \(T_x\ T_y\  \theta(T_x,T_y)\) \textit{customTaper}.

scriptingDiscRingPulleysBezier.cnst (Bezier-angled defined coupling), scriptingDiscRingPulleysBezierLC.cnst (Bezier - coupling length defined) and scriptingDiscRingPulleysArc.cnst (Arcs - both angle and coupling length defined) show various examples of the disc pulley system described in sections 2.9.9.2 to 2.9.9.16.

scriptingDiscRingSymmetricPulleyDOUBLE1.cnst, scriptingDiscRingSymmetricPulleyDOUBLE2.cnst, scriptingDiscRingSymmetricPulleyDOUBLE3.cnst and scriptingDiscRingSymmetricPulleyDOUBLE4.cnst show various examples of the ring-disc systems with two coupling regions described in sections 2.9.9.2 to 2.9.9.16.

scriptingExampleBasicScriptingExample.cnst is an example of a basic script shown in section 2.1.9.1.

scriptingExampleElectrode.cnst is an example script of labeled electrodes with alignment crosses as shown in section 2.1.9.3.

scriptingExampleLabeledCalibrationRuler.cnst example script creates the labeled calibration ruler in section 2.1.9.2.

scriptingExampleMEMSv1.cnst script creates a MEMS flexure with two linear comb drive elements. Details of this scripts are included in section 2.1.9.4.

scriptingExampleMEMSv2.cnst script creates four MEMS radial comb drives connected to a circular hub as described in section 2.1.9.5.

scriptingFractals.cnst shows the use of various fractal constructors defined in section 2.7.3.

scriptingFunctionPlot.cnst shows several functional plotting examples using constructors defined in section 2.7.4.

scriptingGratingsAndGratingCoupler.cnst is an example using constructors from sections 2.9.7.1 and 2.9.7.3.

scriptingGrayscale.cnst is an example using a variety of grayscale constructors from section 2.7.5.

scriptingInstanceExamples.cnst are a collection of examples using the \textit{instance} constructor section 2.4.4.

scriptingLabelMaker.cnst was used to create the GDS rendered output in Figure 2.45.

scriptingLabelOutline.cnst illustrates the use of constructors from section 2.6.4. Here, text outlines are used to generate chip labels.
scriptingLogos.cnst example shows the use of the three logo constructors from section 2.6.6.

scriptingMeanderChannels.cnst illustrates various meandering channels with end reservoirs using constructors from section 2.7.8.

scriptingMEMSactuators.cnst shows various examples of MEMS actuators as described in section 2.10.1.

scriptingMEMSarraysFLAT.cnst and scriptingMEMSarraysHIERARCHY.cnst shows interacting MEMS arrays as described in section 2.10.7.

scriptingMEMScantilevers.cnst shows various examples of cantilevers of varying length extending from a base as described in section 2.10.5.

scriptingMEMSdoublyClampedBeams.cnst shows various examples of doubly clamped beams of varying length extending from a base as described in section 2.10.6.

scriptingMEMSflexures.cnst shows various MEMS accelerometer type anchored flexures with a proof mass as described in section 2.10.4.

scriptingMEMSstressStrainMeasurementStructures.cnst shows various MEMS stress-strain measurement devices including Guckel Rings (see section 2.10.8).

scriptingMISCobjects.cnst shows a variety of miscellaneous objects (interdigitated electrodes) found in section 2.7.

scriptingMultiFile1.cnst, scriptingMultiFile2.cnst, scriptingMultiFile4.cnst and scriptingMultiFile8.cnst are files used to generate scriptingMultiFile.gds. This is an example that calls multiple script files to generate a GDS output as seen in section 2.2.8.

scriptingPhotonicCrystalsHexArrays.cnst example demonstrates the use of photonic crystals defined in section 2.9.8.

scriptingPhotonicsCoupledWaveguideDiscTip.cnst example demonstrates the use of various coupled architectures of waveguides, discs and tips, as defined in section 2.9.15.

scriptingPhotonicsCouplersBendsMisc.cnst example demonstrates the use of various waveguides, slot waveguides and couplers defined in sections 2.9.1.1 to 2.9.3.15.

scriptingPhotonicsGratingCouplersWithWaveguides.cnst example highlights grating couplers with integrated waveguides as defined in section 2.9.7.4.

scriptingPillarHoleHexagonalSquareArrays.cnst example demonstrates the use of pillar-hole hexagonal and square arrays defined in section 2.4.6.

scriptingPoints2Instance.cnst is a script demonstrating structure instantiation at user defined \((x, y)\) point pairs. In this case a structure test is created and instantiated along a spiral path. Coordinates for the path were calcu-
lated using Excel and presented in the supplementary CNSTscripting.xlsx file (SpiralArray sheet).

scriptingPoints2Instance.cnst script renders two user defined shapes. Both shapes are defined within the supplementary CNSTscripting.xlsx file (CustomTaper and Gaussian sheets).

scriptingPolyPath.cnst script creates two distinct paths along user defined \((x,y)\) point pairs. Here, the two paths are defined within the supplementary CNSTscripting.xlsx file (CustomTaper and Gaussian sheets).

scriptingPostScript.cnst demonstrates PostScript file conversion to a GDS rendered shape. Postscript statements were extracted from the supplementary file scriptingPostScript.eps using Figure 2.47 guidelines.

scriptingRaceTrack.cnst was used to create the GDS rendered output in Figure 2.131.

scriptingSpiralDelayLines.cnst describes the usage of spiral delay line constructors (section 2.9.5). The example illustrates the \(S_T\) (number of skipped turns) parameter from the spiralDelayLineArchV2 constructor. In the example, \(S_T\) varies from 0 to 8 while all other parameters remain constant.

scriptingSpiralDelayLinesInverse.cnst similar to the previous example, except the spirals form a slot waveguide of width \(W\) defined by the exposure sleeve \(W_e\). Example describes the inverse spiral delay line constructors (Archimedes and Fermat as defined in section 2.9.6) and illustrates the \(S_T\) (number of skipped turns) parameter from the spiralDelayLineArchV2Inv constructor. In the example, \(S_T\) varies from 0 to 4 while all other parameters remain constant.

scriptingSpirals.cnst demonstrates production of Archimedes, Fermat and Logarithmic spirals as defined in section 2.7.14.

scriptingText.cnst demonstrates the use of textgds constructor for creating text objects within GDS files.

scriptingTextOutline.cnst example shows the use of the textOutline constructor. As shown in figure 2.44, the rendered fonts are defined by a shape outline of a user-specified width (fontOutline). This feature is useful when labeling devices defined by write-time-limited, electron beam lithography systems.

scriptingVerniers.cnst illustrates the use of verniers between two lithographic levels.

scriptingWaveGuidePhCs.cnst example demonstrates the use of various photonic-crystal based waveguides defined in sections 2.9.10 to 2.9.14.
2.2 Interface Functions

2.2.1 Comments

Line comments begins with #. Comments begin with a new line. Comments cannot be appended within or at the end of a scripting command line.

2.2.2 Creating Structures (Cells)

The method creates a GDS structure (cell). Subsequent elements will be placed into the defined cell. If the cell already exists, then elements will be added to the existing structure. Structure names may be up to 32 characters long. Allowed structure character names are A−Z, a−z, 0−9, underscore (_), question mark (?) and dollar sign ($).

```
<structureName    struct>
```

2.2.3 Layer

The method defines a GDS layer for subsequent shapes. Allowed GDS layer numbers range from 0−255. Any value outside this range will automatically be set to GDS layer 0.

```
layerNumber      layer
```

2.2.4 Data Type

The method defines a GDS data-type for each layer. Allowed GDS data-type numbers range from 0−255. Any value outside this range will automatically be set to GDS data-type 0.

```
dataTypeNumber   dataType
```

2.2.5 GDS Rendering Resolution

The method sets the snapping grid of the final GDS output. Default global value is 0.001µm. Parameter gdsResolution is specified in units of micrometers.

```
gdsResolution     gdsReso
0.001            gdsReso defines a 0.001µm rendering resolution.
```
2.2.6 Shape Resolution

The method sets the rendering resolution resolution for subsequent shapes (primarily vector defined shapes, S-Bends, Y-Bends, postscript file conversions, Text elements, etc). Default global value is $0.001\mu m$. This value could be changed for each shape, providing flexibility on the number of points used to define a particular GDS shape. Parameter $shapeResolution$ is specified in units of micrometers.

\[\text{shapeResolution} \quad \text{shapeReso} \]

\[0.001 \quad \text{shapeReso} \quad \text{defines a} \quad 0.001\mu m \text{ rendering resolution.}\]

![Figure 2.16: shapeReso example illustrating rendering a text object at (a) $1\mu m$ (b) $0.1\mu m$ and (c) $0.01\mu m$.](image)

2.2.7 Font Outline Width

The method sets the font outline width (in micrometers). This is used for outline text and label objects (see sections 2.44 and 2.6.4). Default global value is $0.1\mu m$. This value could be changed for each outlined object.

\[\text{fontOutlineWidth} \quad \text{fontOutline} \]

\[0.400 \quad \text{fontOutline} \quad \text{defines a} \quad 0.400\mu m \text{ font outline width.}\]
2.2.8 Calls To Multiple CNST Script Files

The file
data constructor allows script files to call other CNST script files. The called script contents (from file cnstScriptFileName) will be executed until the end of file is reached, then the execution process jumps back to the original script and continues to process shapes. Overall, this method allows users to break up the process flow over many script files.

\[
\text{FileName}
\]

Script files residing in folders within the root directory can be called in the following manner:

Windows:

\[
\text{dirName\cnstScriptFileName
}
\]

Mac/Unix/Linux:

\[
\text{dirName/cnstScriptFileName
}
\]

NOTE: Directories and file names can not contain white characters such as spaces or tabs. This method assumes that cnstScriptFileName has the extension .cnst, i.e. cnstScriptFileName.cnst, hence .cnst must be omitted in the file label cnstScriptFileName. The root directory is defined by the <OpenDirectory> field within the CNSTdefaultValues.xml setup file. Also, backslash (Windows) and forward slash (MacOSX, Linux) characters are omitted prior to the first directory statement (dirName).

2.2.9 Log File Time Date Stamp

At the end of each GDS export a log file, containing a variety of information, including error messages, is created. The log filename assumes the name of the CNST script, with .log extension. During subsequent executions of a distinct script file, the log file will be overwritten. The constructor logFileTimeDate creates log files with filenames containing additional time and date information, hence each script execution will create a new log file.
2.3 Shapes

2.3.1 Circles and Ellipses

2.3.1.1 Primitive Ellipse

Elliptical shape is centered at \((x, y)\), defined by two radii \((r_x, r_y)\), the number of sides \((N_{sides})\) and rotated about the center at an angle \(\theta_{(x,y)}\) expressed in degrees. In this scenario, the shape defining vertices are evenly distributed at angular increments of \(2\pi/N_{sides}\).

\[
x \quad y \quad r_x \quad r_y \quad N_{sides} \quad \theta_{(x,y)} \
\]

ellipse

2.3.1.2 Vectorized Ellipse

Elliptical shape is constructed from Bezier curves, centered at \((x, y)\), defined by two radii \((r_x, r_y)\), and rotated about the center at an angle \(\theta_{(x,y)}\) expressed in degrees. Rendering resolution of the vectorized shape (number of shape vertices) is controlled using the \text{shapeReso} parameter (see section 2.2.6). Unlike the primitive ellipse example where shape vertices are evenly distributed, since the vectorized form is constructed using Bezier curves, vertices are not regularly spaced. More vertices are allocated to regions of higher curvature.

\[
x \quad y \quad r_x \quad r_y \quad \theta_{(x,y)} \
\]

ellipseVector

Figure 2.17: \textit{ellipse} and \textit{ellipseVector} constructor parameters.
2.3.2 Circle through three points

The resulting shape is defined by the three points and the number of sides ($N_{sides}$).

![Diagram of a circle with three points](image)

**Figure 2.18:** `circlethree` parameters.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$y_1$</th>
<th>$x_2$</th>
<th>$y_2$</th>
<th>$x_3$</th>
<th>$y_3$</th>
<th>$N_{sides}$</th>
<th><code>circlethree</code></th>
</tr>
</thead>
</table>
2.3.3 Circle With a Wave Boundary

Circular structure of radius $r$ with a sinusoidal boundary centered at $(x, y)$. $A$ represents the sine wave amplitude, $n$ is an integer number of oscillations along the boundary extending from 0 to $2\pi$, $N_s$ is the number of sides used to construct the boundary, and $\theta_{(x,y)}$ is the rotation about the center point.

$x \ y \ r \ n \ A \ N_s \ \theta_{(x,y)} \ circleWave$

Figure 2.19: Circle with a sinusoidal boundary variation.
2.3.4 Cross

Cross is defined by the center \((x, y)\), width \((W)\), length \((L)\) and rotation about the center at an angle \(\theta_{(x,y)}\) expressed in degrees.

\[
\begin{array}{cccccc}
 x & y & W & L & \theta_{(x,y)} & \text{cross}
\end{array}
\]

![Diagram of cross constructor parameters](image)

Figure 2.20: cross constructor parameters.
2.3.5 L-Shape

L-shape is defined by the center \((x, y)\), widths \((W_1\) and \(W_2\)), lengths \((L_1\) and \(L_2\)) and rotation about the center at an angle \(\theta_{(x,y)}\) expressed in degrees.

\[
\begin{array}{cccccccc}
    x & y & W_1 & L_1 & W_2 & L_2 & \theta_{(x,y)} & \text{Lshape} \\
\end{array}
\]

Figure 2.21: \textit{Lshape constructor parameters.}
2.3.6 Pie Shaped Arc

Arc shape is centered at \((x, y)\), defined by two radii \((r_x, r_y)\), sweep angles, the number of sides \((N_{sides})\) and rotated about the center point by \(\theta_{(x,y)}\). Sweep angles, start and end angles \((\theta_s, \theta_e)\), and \(\theta_{(x,y)}\) are expressed in degrees.

\[
\begin{align*}
  x & \quad y & \quad r_x & \quad r_y & \quad \theta_s & \quad \theta_e & \quad N_{sides} & \quad \theta_{(x,y)} & \quad \text{arc}
\end{align*}
\]

Figure 2.22: arc constructor used to create user defined pie-shaped arc sections.

2.3.7 Pie Shaped Arc - Vector

The vectorized pie shaped arc is constructed using Bezier curves. Rendering resolution of the defined shape is controlled using the \texttt{shapeReso} parameter defined in section 2.2.6.

\[
\begin{align*}
  x & \quad y & \quad r_x & \quad r_y & \quad \theta_s & \quad \theta_e & \quad \theta_{(x,y)} & \quad \text{arcVector}
\end{align*}
\]

Figure 2.23: Illustration shows an arc section at one end of a waveguide. Rounding waveguide corners alleviates stress crowding at sharp corners. This is particularly useful when patterning thick stressed layers (resist, thin films, etc).
2.3.8 Polygon

Polygon is defined by the center \((x, y)\), outer radius \((r)\), number of sides \((N_{\text{sides}})\), and rotation about the center at an angle \(\theta_{(x,y)}\) expressed in degrees. A shape is characterized by an arbitrary number of vertices to form a closed figure.

\[
x \quad y \quad r \quad N_{\text{sides}} \quad \theta_{(x,y)} \quad \text{polygon}
\]

![Figure 2.24: polygon constructor parameters.](image_url)
2.3.9 Rectangle

The shape consists of four 90° corners. Three constructors are used to define the rectangular object.

\[ x_{LL} \quad y_{LL} \quad x_{UR} \quad y_{UR} \quad \theta_{(x_{LL},y_{LL})} \quad \text{rectangle} \]

\[ x_{LL} \quad y_{LL} \quad L \quad H \quad \theta_{(x_{LL},y_{LL})} \quad \text{rectangleLH} \]

\[ x_{C} \quad y_{C} \quad L \quad H \quad \theta_{(x_{C},y_{C})} \quad \text{rectangleC} \]

Figure 2.25: Three rectangle constructors. (a) rectangle, (b) rectangleLH, and (c) rectangleC
2.3.10 Rounded Rectangle

Rounded rectangle constructor \texttt{roundrect} is defined using the lower left corner \((x_{LL}, y_{LL})\), length \((L)\) and height values \((H)\). The curved sections of the rounded rectangle are defined by the two radii \((r_x, r_y)\) and are rendered using the specified \texttt{shapeReso} resolution parameter (see section 2.2.6). Rotation of the rounded rectangle is about the lower left corner \((x_{LL}, y_{LL})\). Rounded rectangles defined using \texttt{roundrectC} are of length \(L\), height \(H\) and are centered at \((x_C, y_C)\).

\[
\begin{align*}
    x_{LL} & \quad y_{LL} & \quad L & \quad H & \quad r_x & \quad r_y & \quad \theta_{(x_{LL}, y_{LL})} & \text{roundrect} \\
    x_C & \quad y_C & \quad L & \quad H & \quad r_x & \quad r_y & \quad \theta_{(x_C, y_C)} & \text{roundrectC}
\end{align*}
\]

**Figure 2.26:** Rounded rectangle constructors defined by (a) lower left corner \((\text{roundrect})\) and (b) center coordinates \((\text{roundrectC})\).
2.3.11 Rectangular SU-Shape

The below constructor creates a rectangular S- and U-shape objects. Positive and negative values of the three lengths \((L_i)\) allow for a variety of S- and U-shapes. These structures are useful for routing metal runners to bond pads.

\[
x \quad y \quad L_1 \quad L_2 \quad L_3 \quad W \quad \theta_{(x,y)} \quad \text{rectSUshape}
\]

Figure 2.27: Rectangular SU-shapes constructed using various length values. (a) all lengths \(L_i\) are positive, (b) \(L_1, L_3\) positive, \(L_2\) negative, (c) \(L_1, L_2\) positive, \(L_3\) negative and (d) \(L_1, L_2\) negative, \(L_3\) positive values.
2.3.12 Rectangle With a Linear Taper

Below constructor creates a parametrized rectangle of width $w_1$ and length $L_1$ connected to a taper. Tapered region varies linearly over the taper length $L_2$ from a the rectangle width $w_1$ to the end taper width $w_2$.

$$x \ y \ w_1 \ L_1 \ w_2 \ L_2 \ \theta_{(x,y)} \ \text{rectTaper}$$

Figure 2.28: Rectangle with a taper at one end.
2.3.13 Star

Star is defined by the center \((x, y)\), the two radii \((r_i\) and \(r_o\)), number of points \((N_{\text{points}})\), and rotation about the center at an angle \(\theta_{(x,y)}\) expressed in degrees.

\[
x \quad y \quad r_i \quad r_o \quad N_{\text{points}} \quad \theta_{(x,y)} \quad \text{star}
\]

Figure 2.29: \textit{star} constructor parameters.
2.3.14 Torus - Arc

A shape defined by torus is centered at \((x, y)\), defined by an inner and outer radii \((r_i, r_o)\), sweep angle, and the number of sides \((N_{\text{sides}})\). Sweep angles, start and end angles \((\theta_s, \theta_e)\) are expressed in degrees. The torusW constructor creates a similar shape defined by the midpoint radius \(r\) and width \(w\).

![Figure 2.30: Example of arc sections defined by (a) torus and (b) torusW constructors.](image)

\[
\begin{align*}
x & \quad y & \quad r_i & \quad r_o & \quad \theta_s & \quad \theta_e & \quad N_{\text{sides}} & \quad \text{torus} \\
x & \quad y & \quad r & \quad w & \quad \theta_s & \quad \theta_e & \quad N_{\text{sides}} & \quad \text{torusW}
\end{align*}
\]

2.3.15 Torus - Vector

The vectorized torus is centered at \((x, y)\) and defined by an inner and outer radii \((r_i, r_o)\). The shape is constructed by subtracting two vectorized circles (vectorized ellipses with \(r_x = r_y\), see section 2.3.1.2). Rendering resolution of the defined shape is controlled using the shapeReso parameter defined in section 2.2.6.

\[
x \quad y \quad r_i \quad r_o \quad \text{torusVector}
\]
2.3.16 Torus With a Wave boundary

Torus structure with a sinusoidal boundary of inner and outer radii \( r_i \) and \( r_o \). The structure is centered at \((x, y)\) with inner and outer sinusoidal boundaries in phase or \( \frac{\pi}{2} \) out of phase. \( A \) represents the sine wave amplitude, \( n \) the number of oscillations along the inner and outer boundaries extending from 0 to 2\( \pi \), \( N_s \) is the number of sides used to construct each boundary, and \( \theta_{(x,y)} \) is the rotation about the center point.

\[
\begin{align*}
  x & \quad y & \quad r_i & \quad r_o & \quad n & \quad A & \quad N_s & \quad \theta_{(x,y)} & \text{torusWaveIn} \\
  x & \quad y & \quad r_i & \quad r_o & \quad n & \quad A & \quad N_s & \quad \theta_{(x,y)} & \text{torusWaveOut}
\end{align*}
\]

Figure 2.31: Torus wave with inner and outer radii (a) in phase (torusWaveIn) and (b) \( \frac{\pi}{2} \) out of phase (torusWaveOut).
2.4 Arrays and Instances

2.4.1 Rectangular Arrays

The method instantiates and arrays a GDS structure on a periodic rectangular grid. 2 constructors are available. Both define the starting point of the array \((x, y)\) along with the number of columns \((N_{columns})\) and rows \((N_{rows})\). First (denoted as the 1 parameter prior \(arrayRect\) constructor) has a defined pitch of \(\Delta x\) and \(\Delta y\) in the \(x\) and \(y\) directions respectively. The second (denoted as the 2 parameter prior \(arrayRect\) constructor) is characterized by the number of elements distributed evenly between the start \((x, y)\) and end \((x_e, y_e)\) points of the array.

\[
\begin{align*}
\Delta y \\
(x, y) \quad \Delta x
\end{align*}
\]

\[
\begin{align*}
\Delta y \\
(x, y) \quad (x_e, y_e)
\end{align*}
\]

(a) \hspace{1cm} (b)

Figure 2.32: Two versions of the rectangular array constructor \(arrayRect\) (a) 1 (b) 2.

- \(<structureToBeArrayed x y N_{columns} N_{rows} \Delta x \Delta y 1 \ arrayRect>\)
- \(<structureToBeArrayed x y N_{columns} N_{rows} x_e y_e 2 \ arrayRect>\)
2.4.2 Hexagonal Arrays

This method arrays GDS structures on a periodic hexagonal grid. The resulting arrayed structure consists of two rectangular arrays. Hexagonal arrays are characterized by the starting coordinate \((x, y)\), number of columns and rows \((N_{\text{columns}} \text{ and } N_{\text{rows}})\) are specified, with previously defined structure instantiated on a hexagonal grid with spacing defined by \(\Delta s\).

![Hexagonal Array Diagram](image)

**Figure 2.33:** arrayHex constructor used to create hexagonal arrays.

\[
<\text{structureToBeArrayed} \ x \ y \ N_{\text{columns}} \ N_{\text{rows}} \ \Delta s \ \text{arrayHex}>
\]
2.4.3 Polar Arrays

The array method instantiates GDS structures in polar coordinates. There are 2 main versions of the array polar constructor, each with two variants. Versions 1 and 1R: Start ($\theta_s$) and end angles ($\theta_e$), along with start ($r_s$) and end ($r_e$) radii, are specified, with a previously defined structure instantiated at angular and radial increments defined respectively by $\Delta \theta$ and $\Delta r$. In figure 2.34a, GDS structure named $structureToBeArrayed$ is instantiated starting angularly at $\theta_s = 60$ to $\theta_e = 120$, in angular increments of 30 degrees, and radially starting at $r = 2.0 \mu m$ to $r = 4.0 \mu m$ with radial increments of 1.0 $\mu m$. Version 1R rotates each instance to point towards $r = 0$.

Versions 2 and 2R: Start ($\theta_s$) and end angles ($\theta_e$), along with start ($r_s$) and end ($r_e$) radii, are specified, number of instances between the defined regions is defined by $\theta#_{\theta}$ and $r#_{r}$ in the angular and radial directions respectively. Region of space defined by the start and end parameters is uniformly subdivided. In figure 2.34b, structure $structureToBeArrayed$ is instantiated $\theta#_{\theta} = 5$ times starting angularly at $\theta_s = 60$ to $\theta_e = 120$, and radially $r#_{r} = 5$ times starting at $r = 2.0 \mu m$ to $r = 4.0 \mu m$. Version 2R rotates each instance to point towards $r = 0$.

![Figure 2.34](image)

Figure 2.34: Four versions of the polar array constructor arrayPolar. (a) 1 and 1R, (b) 2 and 2R.
Figure 2.35: Polar Array example. (a) cell containing two objects, versions (b) 1 and (c) 1R of the arrayPolar constructor.

2.4.4 Instancing GDS Structures

The method instantiates GDS structures at a point \((x, y)\) within other structures. The resulting instances are characterized by mirror, scaling and rotation parameters. Mirroring parameter MIR assumes values of \(X\) and \(Y\) for symmetric reflections around respective axes. Any other MIR values represent unmirrored structures (i.e. using a single character other than \(X\) and \(Y\), i.e. \(N\)). Magnification (MAG) and rotation (\(\theta\)) are double values.

\[
\text{<structureToBeInstanced } x \ y \ MIR \ MAG \ \theta \ \text{instance>}
\]

Below constructor first symmetrically centers the structure around the origin by employing user-defined pattern boundary parameters then transforms the instantiated structure. The bounding box is defined by the lower left \((x_{LL}, y_{LL})\) and upper right \((x_{UR}, y_{UR})\) coordinates of the structural boundary. This method could be useful when instantiating various structures for a multi-image stepper reticle design. Transformation parameters, translation \((x, y)\), mirroring (MIR), magnification (MAG) and rotation (\(\theta\)) are defined in an identical manner as for the instance constructor.

\[
\text{<structureToBeInstanced } x \ y \ x_{LL} \ y_{LL} \ x_{UR} \ y_{UR} \ MIR \ MAG \ \theta \ \text{instanceSym>}
\]
2.4.5 Points To Instance

This method instantiates GDS structures along a list of user defined points. This methodology is accomplished within the following two step process: first, by defining the instanced structure in double brackets, then instatiating the structure at the specified coordinates, terminated by the constructor `points2instance`.

![Diagram of points to instance](image)

**Figure 2.36:** `points2instance` constructor used to instantiate structures (cells) at user defined \((x_i, y_i)\) positions.

```
«structureToBeInstanced»
\[ x_1 \quad y_1 \quad x_2 \quad y_2 \quad \cdots \quad x_n \quad y_n \]
points2instance
```

**Figure 2.37:** Points to instance example. Structure instanced along a path defined by a set of \((x, y)\) coordinate pairs.
2.4.6 Pillar-Hole Hexagonal and Square Arrays

This method constructs hexagonal and square lattice \((N_x \times N_y)\) arrays of periodic structures. Number of vertex points \(N_{\text{side}}\) defines the polygon, i.e. \(N_{\text{side}} = 8\) defines an octagon. Vertices of the vectorized shapes are defined by the \textit{shapeReso} parameter. Vectorized array constructors are defined by the suffix \textit{V}. Narrow elliptical shapes are best defined in vectorized forms. In this case, the curved section comprises a dense distribution of vertices. Each rendered shape has a rotational degree of freedom controlled by the \(\theta\) parameter. Array positions are defined either by the center of the lower-left shape or by the array centroid (defined by the constructor suffix \textit{C}).

Hexagonal and square pillar-hole arrays:

\[
\begin{align*}
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrPillar>} \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexPillar}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrHole}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexHole}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrPillarC}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexPillarC}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrHoleC}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad N_{\text{side}} \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexHoleC}> \\
\end{align*}
\]

Vectorized hexagonal and square pillar-hole arrays:

\[
\begin{align*}
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrPillarV}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexPillarV}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrHoleV}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexHoleV}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrPillarVC}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexPillarVC}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad p_y \quad N_x \quad N_y \quad \theta & \quad \text{sqrHoleVC}> \\
\text{<uniqueStructName} & \quad x \quad y \quad r_x \quad r_y \quad p_x \quad N_x \quad N_y \quad \theta & \quad \text{hexHoleVC}> \\
\end{align*}
\]
Figure 2.38: Square and hexagonal arrays of holes defined by the lower-left corner 
((a) and (b)) and by the centroid ((c) and (d)).

Figure 2.39: Various pillar-hole examples. (a) $N_{\text{side}} = 4$, (b) $N_{\text{side}} = 4$ and $\theta = 45$ degrees, (c) ellipses with $N_{\text{side}}$ defined, (d) vectorized ellipses with denser number of vertices at curved sections where shapeReso defines the rendered resolution, (e) ellipse array with $\theta = 22$ degrees.
2.5 General Area Operations

2.5.1 Boolean Operations

Constructive area geometry allows for the creation of complex shapes by means of Boolean operators between generalized areas. The processing strategy is to create objects within a variety of GDS structures. A collection of objects residing in a particular GDS Layer (extractedGDSLayer) are then extracted from an existing structure. Using the genArea constructor, the objects are then stored into a generalized area defined by a distinct string (genAreaName). This operation is repeated until two or more distinct generalized areas are created. Boolean operation between two generalized areas is then carried out in order to create desired compound geometries. The results of the Boolean operation are cast into a user defined GDS layer within an initialized GDS structure. Figure 2.40 illustrates the AND, OR, SUBTRACT and XOR operations between two general areas (A and B). Additionally, as shown in the following section (2.5.2), an affine transformation could be implemented with genArea objects.

![Figure 2.40: Boolean operations between two generalized area objects. (a) two general area objects A and B, (b) A (AND) B, (c) A (OR) B, (d) A (SUBTRACT) B, (e) A (XOR) B.](image)

The following constructor extracts shapes residing in a GDS layer, from an existing GDS structure, and then stores the contents into a general area object. genAreaName is a string constructed from a continuous set of characters without spaces or tabs.

```
<genAreaName structName extractedGDSLayer genArea>
```

Boolean operations between two general area objects:

```
<genAreaName1 genAreaName2 resultGDSLayer OPERATION>
```

OPERATION constructor takes on the following values: AND, OR, SUBTRACT and XOR. Results from the operations are stored into a currently initialized GDS structure.
IMPORTANT NOTES:
1) genAreaName string identifier can not contain spaces or tabs.
2) When extracting large instantiated GDS arrays, the genArea constructor flattens the data (i.e. the GDS structure hierarchy is not preserved). Similar to Boolean operations, the procedure is computationally intensive when large instantiated arrays are used. Consequently, these operations could take a considerable amount of time. For large periodic arrays, built using the instance constructor, the above process is considerably faster when operations are performed on a smaller instantiated GDS structure. This process circumvents a host of dilemmas and preserves the hierarchy of the arrayed GDS structure.
3) Processing large, clear-field arrayed patterns could lead, under certain operations, to shapes with a large number of vertices. Some commercial packages have an upper vertex limit per GDS shape.

Boolean SUBTRACT example:

```xml
<devices struct>
  # create an ellipse in layer 11
  11 layer
  0 0 2 4 44 0 ellipse
  # create a pentagon in layer 7
  7 layer
  1 0 1.5 1.5 5 0 ellipse

  # extract shapes in layer 11 and store into variable genArea1
  <genArea1 devices 11 genArea>

  # extract shapes in layer 7 and store into variable genArea2
  <genArea2 devices 7 genArea>

  # create struct to store results
  # genArea1 SUBTRACT genArea2 cast to GDS layer 44
  <resultsSUBTRACT struct>
    <genArea1 genArea2 44 subtract>
```

Figure 2.41: Boolean SUBTRACT operation between an ellipse and a pentagon. (a) CNST script, (b) ellipse and a pentagon GDS shapes, (c) resulting GDS shape following a Boolean subtraction between an ellipse and a pentagon.
2.5.2 General Area Copies, Shape Bias and Affine Transformations

This module copies general areas into a currently initialized GDS structure. The general areas are extracted using the `genArea` constructor. The objects then undergo affine transformations (translation, mirroring, scaling, rotation and shape bias). Resulting areas are then cast into a GDS layer and copied into a currently initialized GDS structure. The utility of this option is manifested in myriad forms, especially when casting multiple stepper images onto a single reticle from a multi-layer GDS structure design. Also, unlike instantiation, where layer properties are inherited from the parent structure, this module allows for users to define GDS layer numbers for each general area copy.

Variable `genAreaName` defines the areas extracted via `genArea`, `x` and `y` are translation directions in micrometers, `MIRROR` parameter is used to horizontally (around y-axis) or vertically (around x-axis) flip (mirror) the areas, `MAG` defines the pattern scaling, pattern rotation is defined by `θ`, and `resultGDSLayer` defines the GDS layer of the resulting general area shapes. The `MIRROR` parameter assumes values of X and Y for symmetric reflections around respective axes. Any other `MIRROR` values represent unmirrored structures (i.e. using a single character other than X and Y, i.e. N). `MAG` is a positive double value indicating area magnification. `MAG = 1` omits area scaling. Rotation defined by the parameter `θ` is expressed in degrees.

`BIAS` parameter defines the amount of shape perimeter expansion (positive bias) or contraction (negative bias) in micrometers (see Figure 2.42).

The module supplies the following two constructors:
- `genAreaCopy` - translates objects relative to the structural coordinates of the general area objects.
- `genAreaCopyC` - first centers the object around the origin, then performs affine transformations, i.e. translation is with respect to to `(0, 0)`.

```
<genAreaName x y MIRROR MAG θ BIAS resultGDSLayer genAreaCopy>
<genAreaName x y MIRROR MAG θ BIAS resultGDSLayer genAreaCopyC>
```

These constructors copy a defined general area (flattened area) into an initialized GDS structure. Unlike `struct` instantiation, the process of copying areas does not retain hierarchy. Consequently the process results in larger files (see Important Notes comment in previous section 2.5.1).
Figure 2.42: Shape biasing (a) expanding and (b) contracting an area using a positive and negative bias.
2.6 Text Labels, PostScript and Logos

2.6.1 Text

Text shape is characterized by a string of characters using vector based system fonts. Text shapes are directly placed within the active GDS structure. There are two constructors for text shapes, where placement is either at the lower left corner (textgds) or at center (textgdsC) of the text string. Interface function parameter shapeReso defines the rendering resolution of the text. Parameter someText represents all printable characters. If the fontName does not exist, "Serif" will be used and a comment will be placed within the error log file. If "Serif" does not exist, then first encountered font within the system font list will be used. (x, y) values are in micrometers. Font size is specified by conventional fontmetrics definitions, measured from the descender to the ascender line in micrometers.

```
<{{someText}}>{{fontName}} fontSize x y textgds>
<{{someText}}>{{fontName}} fontSize x y textgdsC>
```

![Figure 2.43: GDS rendered textgds example.](image)

The following are scripts used to generate the text in Figure 2.43:

```
<{{SomeText!@#$}}>{{Algerian}} 20 0 0 textgds>
<{{4!AaBbCc}}>{{Serif}} 20 0 0 textgds>
<{{4!AaBbCc}}>{{Arial}} 20 0 0 textgds>
```
2.6.2 Text Outline

Text outline shapes are useful when lithographic write times (e.g. electron beam lithography) are of consideration. In a similar manner to Text shape in the previous section, the outline shape is characterized a string of text, font name and size, width of the outline and position \((x, y)\). There are two constructors for text shapes, where placement is either at the lower left corner (textOutline) or at center (textOutlineC) of the text string. Interface function parameter shapeReso defines the rendering resolution of the text. If the fontName does not exist, "Serif" will be used and a comment will be placed within the error log file. If "Serif" does not exist, then first encountered font within the system font list will be used. \((x, y)\) values are in micrometers. Font size is specified by conventional fontmetrics definitions, measured from the descender to the ascender line in micrometers. Outline width is specified using the fontOutline parameter (see section 2.2.7).

\[
<\{(\text{someText})\} \{(\text{fontName})\} \text{fontSize} \ x \ y \ \text{textOutline}>
\]

\[
<\{(\text{someText})\} \{(\text{fontName})\} \text{fontSize} \ x \ y \ \text{textOutlineC}>
\]

Figure 2.44: Text Outline GDS example.

IMPORTANT NOTE: Within this object, errors dealing with exceeding maximum number of GDS vertices could be encountered. To circumvent this dilemma, when using larger letters, increase the shapeReso parameter.
2.6.3 Label Maker

Label maker has 6 available constructors. 4 constructors automatically generate labels based on the number of rows and columns. Chip labels are either numbers (autoOuter and autoRowColumn) or a combination of numbers and letters (autoOuterLetters and autoRowColumnLetters). 2 remaining constructors generate custom, user-defined labels (outer and rowColumn). Each constructor pair has an outer and row-column option. Label placement for the outer option is along the top and left side of the chip array. Top side placement starts at \((x,y)\) and the left hand side placement initiates at \((x_r,y_r)\). Label placement for the row-column option initiates with the top-left label at a position \((x,y)\). In both cases number of rows \((N_{row})\) and columns \((N_{col})\), font name \((fontName)\), font size \((fontSize\text{ in } \mu m)\), and the pitch along the two directions \((\Delta_x\text{ and } \Delta_y)\) are specified. Similarly, custom labels are defined by an additional set of label parameters \((L_1, L_2 \ldots L_n)\). In all cases, if the font name is not contained within the system font list, label maker will default to a Serif font.

4 auto labels:

\begin{verbatim}
{N_{row} N_{col} fontName fontSize x y x_r y_r \Delta_x \Delta_y autoOut labelMaker}
{N_{row} N_{col} fontName fontSize x y x_r y_r \Delta_x \Delta_y autoRowCol labelMaker}
{N_{row} N_{col} fontName fontSize x y x_r y_r \Delta_x \Delta_y autoOutLett labelMaker}
{N_{row} N_{col} fontName fontSize x y x_r y_r \Delta_x \Delta_y autoRowColLett labelMaker}
\end{verbatim}

2 custom labels where parameters are TAB SEPARATED:

\begin{verbatim}
{L_1 L_2 \ldots L_n N_{row} N_{col} fontName fontSize x y x_r y_r \Delta_x \Delta_y out labelMaker}
{L_1 L_2 \ldots L_n N_{row} N_{col} fontName fontSize x y x_r y_r \Delta_x \Delta_y rowCol labelMaker}
\end{verbatim}

**IMPORTANT NOTES:**

1) Labels \(L_1\ldots L_n\) are constructed from any printable ASCII character, including the space character. Therefore, space separation between constructor parameters is NOT allowed. All specified parameters within above constructors must be TAB separated.

2) Carriage returns cannot be used within the script line constructor.
Figure 2.45: Label Maker Constructor Examples: (a) autoOut, (b) autoRowCol, (c) autoOutLett, (d) autoRowColLett, (e) out, and (f) rowCol.
2.6.4 Label Maker - Outline Text

This method is identical to the previously defined label maker section (2.6.3) with the exception that rendered shapes are outlined text objects. Outline width is specified using the `fontOutline` parameter (see section 2.2.7).

4 auto labels:

\[
\begin{array}{c}
N_{row} \ N_{col} \ \text{fontName} \ \text{fontSize} \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOut} \ \text{labelOutline} \\
\end{array}
\]

\[
\begin{array}{c}
N_{row} \ N_{col} \ \text{fontName} \ \text{fontSize} \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowCol} \ \text{labelOutline} \\
\end{array}
\]

\[
\begin{array}{c}
N_{row} \ N_{col} \ \text{fontName} \ \text{fontSize} \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOutLett} \ \text{labelOutline} \\
\end{array}
\]

\[
\begin{array}{c}
N_{row} \ N_{col} \ \text{fontName} \ \text{fontSize} \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowColLett} \ \text{labelOutline} \\
\end{array}
\]

2 custom labels where parameters are TAB SEPARATED:

\[
\begin{array}{c}
L_1 \ L_2 \ldots \ L_n \ N_{row} \ N_{col} \ \text{fontName} \ \text{fontSize} \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{out} \ \text{labelOutline} \\
\end{array}
\]

\[
\begin{array}{c}
L_1 \ L_2 \ldots \ L_n \ N_{row} \ N_{col} \ \text{fontName} \ \text{fontSize} \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{rowCol} \ \text{labelOutline} \\
\end{array}
\]

IMPORTANT NOTES: 1) Labels $L_1 \ldots L_n$ are constructed from any printable ASCII character, including the space character. Therefore, space separation between constructor parameters is **NOT** allowed. All specified parameters within above constructors must be TAB separated. 2) Carriage returns cannot be used within the script line constructor. 3) Within this object, errors dealing with exceeding maximum number of GDS vertices could be encountered. To circumvent this dilemma, when using larger letters, increase the `shapeReso` parameter.
Figure 2.46: Label Maker Outlined Text Constructor Examples: (a) autoRowColLett and (b) rowCol.
2.6.5 PostScript to GDS

2.6.5.1 PostScript Pixel Value

PostScript shapes are defined by double values in conjunction with constructors (moveTo (m), lineTo (l), curveTo (c), etc). The pixel value defines scaling of these doubles. Default global value is 1.000. This implies that the postscript coordinate values are unscaled and in units of micrometers. Parameter \textit{pixelScaling} represents the scaling of the postscript coordinate values. For instance,

\begin{verbatim}
10.000 psPixelValue
\end{verbatim}

scales each postscript value by 10 times. Alternatively,

\begin{verbatim}
0.100 psPixelValue
\end{verbatim}

reduces the postscript image by 10. Once the value is set, subsequent postscript shapes are rendered to this value. This value could be changed for each postscript shape.

\begin{verbatim}
pixelScaling psPixelValue
\end{verbatim}

2.6.5.2 PostScript Fracturing

The following parameter defines the number of fractured postscript shapes. Default global value is 1, implying that the shape is unfractured. The \textit{psFracElements} parameter allows direct control of number of vertices per GDS shape. This is important since aggressively scaled, large, continuous postscript shapes could exceed the maximum number of vertices per GDS shape.

\begin{verbatim}
20 psFracElements
\end{verbatim}

implies that the overall postscript shape area will be subdivided into 20 segments along the y-direction. This value could be changed for each postscript shape.

\begin{verbatim}
numberOfFracturedElements psFracElements
\end{verbatim}

2.6.5.3 Defining Postscript Shapes

Bitmap images could be vectorized using either Illustrator, Inkscape (open source package) or some other package capable of exporting EPS files. Figure 2.47 illustrates the relevant portion of the EPS file. This information is copied and pasted within the CNST script file.
Figure 2.47: *PostScript* scripting `PostScript.eps` example. The arrow denotes the relevant text to be copied and pasted into the CNST script file.

Figure 2.48: *GDS* shape of the rendered *postscript* shape.
2.6.6 CNST and NIST logos

Three constructors are available for placement of CNST and NIST logos. Individual (cnstEmblem, cnstLogo, nistLogo) or combined (nistCnstLogo) are placed at coordinates that define the centroid of the GDS logo shape. Since logos are cast using postscript values, the shapeReso parameter defines the rendering resolution of the logo shapes. Scaling of the resulting logo is set using the scale parameter.

\[
\begin{align*}
    x & \quad y & \quad scale & \quad cnstEmblem \\
    x & \quad y & \quad scale & \quad cnstLogo \\
    x & \quad y & \quad scale & \quad nistLogo \\
    x & \quad y & \quad scale & \quad nistCnstLogo \\
\end{align*}
\]

Figure 2.49: CNST logos created using the (a) \texttt{cnstLogoEmblem} and (b) \texttt{cnstLogo} constructors.

Figure 2.50: NIST logo created using the \texttt{nistLogo} constructor.

Figure 2.51: Combined NIST and CNST logos created using the \texttt{nistCnstLogo} constructor.
2.7 Objects

2.7.1 Arc (Torus-Circle) bounded square-hex arrays

The following constructors create a square or hexagonal array of dots inside of an arc boundary defined by an inner and outer radii ($r_i$ and $r_o$), start and end angles ($\theta_s$ and $\theta_e$) and the number of vertices ($N$). For a torus defined by $r_i$ and $r_o$, $\theta_s = 0$ and $\theta_e = 360$. A circular boundary is represented by a radius $r_o$ with $r_i = 0$, $\theta_s = 0$ and $\theta_e = 360$. The circular dots of radius $r_s$ are either defined by a number of vertices $n_s$ ($\text{arcSquareFill}$ and $\text{arcHexFill}$) or by the $\text{shapeReso}$ parameter for vectorized shapes ($\text{arcSquareFillV}$ and $\text{arcHexFillV}$). Parameter $\theta(x,y)$ defines the shape rotation about the arc origin $(x,y)$.

\begin{verbatim}
<uniqueStructName x y r_i r_o \theta_s \theta_e N \theta(x,y) r_s n_s \Delta x \text{arcSquareFill}>
<uniqueStructName x y r_i r_o \theta_s \theta_e N \theta(x,y) r_s \Delta x \text{arcSquareFillV}>
<uniqueStructName x y r_i r_o \theta_s \theta_e N \theta(x,y) r_s n_s \Delta x \text{arcHexFill}>
<uniqueStructName x y r_i r_o \theta_s \theta_e N \theta(x,y) r_s \Delta x \text{arcHexFillV}>
\end{verbatim}

![Figure 2.52: Square array inside (a) an arc, (b) circular and (c) hexagonal ($N = 8$) boundary. Hexagonal array of dots inside (d) an arc, (e) circular and (f) hexagonal ($N = 8$) boundary.](image)
2.7.2 Bezier Curve

The toolbox employs Bezier curves to represent complex curved objects. The shapes are rendered to a resolution defined by the `shapeReso` parameter. In general, Bezier curve is a parametrized curve used as a path in computer vector graphics. Concatenation of multiple Bezier paths can be used to model smooth curves of arbitrary complexity. A Bezier curve is mathematically expressed as

$$B(t) = \sum_{k=0}^{n} C_k (1-t)^{n-k} t^k P_k$$  \hspace{1cm} (2.1)

where $n$ is the polynomial degree, $k$ is the index, $t$ is a variable ranging $0 \leq t \leq 1$, and $n C_k$ is the binomial coefficient given by

$$n C_k = \frac{n!}{k!(n-k)!}$$  \hspace{1cm} (2.2)

The nanolithography toolbox employs cubic Bezier curves for creating complex curved shapes. With $n = 3$ equation 2.1 becomes

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$  \hspace{1cm} (2.3)

where the corresponding start and end points are $P_0$ and $P_3$, with respective control points $P_1$ and $P_2$. Equation 2.3 for the two coordinates is written as

$$B(t)_x = (1-t)^3 P_{0x} + 3(1-t)^2 t P_{1x} + 3(1-t) t^2 P_{2x} + t^3 P_{3x}$$  \hspace{1cm} (2.4)

$$B(t)_y = (1-t)^3 P_{0y} + 3(1-t)^2 t P_{1y} + 3(1-t) t^2 P_{2y} + t^3 P_{3y}$$  \hspace{1cm} (2.5)

Figure 2.53 shows rendered curves at varying resolution. Rendered curves exhibit an increased vertex density at higher curvatures automatically. The points are uniformly spaced in $t$, but become closer together in $(x,y)$ space as curvature increases.

![Figure 2.53: Cubic Bezier curves rendered at a (a) lower and (b) higher resolution. Dots represent rendered curve vertices.](image-url)
Bezier Curve is defined using a start \((x_1, y_1)\) and end \((x_2, y_2)\) points, two control points \((c_{x_1}, c_{y_1})\) and \((c_{x_2}, c_{y_2})\), curve width \(W\) and \(\theta_{(x_1,y_1)}\) rotation about \((x_1, y_1)\). The \texttt{shapeReso} parameter defines the rendering resolution of the Bezier curve.

\[
\begin{align*}
<x_1 & y_1 c_{x_1} c_{y_1} c_{x_2} c_{y_2} x_2 y_2 W \theta_{(x_1,y_1)} \texttt{bezierCurve}> \\
(x_{1},y_{1}) & (c_{x_{2}},c_{y_{2}}) \quad (x_{2},y_{2}) \\
(c_{x_{1}},c_{y_{1}})
\end{align*}
\]

**Figure 2.54:** Bezier Curve example showing the start and end points along with the corresponding control points.

**Figure 2.55:** Two example GDS shapes of the \texttt{bezierCurve} constructor. Highlighted periphery shows curved sections with densely packed vertices.
Figure 2.56: Schematic illustration showing various parameters from the \texttt{bezierCurveln} constructor.
Figure 2.57: Schematic illustration of the `bezierCurvelnSlot` constructor.
2.7.3 Fractals

Several constructor methods are available for creating self-similar structures, including Sierpinski triangle and carpet, Vicsek saltire and cross, curved trees and various other tree-like structures. The last GDS structure iteration constructed by the recursive fractal generator is instantiated at \((x, y)\). Iteration numbers are postfix appended to the \textit{shortStructName} parameter.

\[
<\text{shortStructName} \ x \ y \ iterations \ Length \ \text{sierpinskiTriangle}>
\]
\[
<\text{shortStructName} \ x \ y \ iterations \ Length \ \text{sierpinskiCarpet}>
\]
\[
<\text{shortStructName} \ x \ y \ iterations \ Length \ \text{vicsekSaltire}>
\]
\[
<\text{shortStructName} \ x \ y \ iterations \ Length \ \text{vicsekCross}>
\]
\[
<\text{shortStructName} \ x \ y \ iterations \ Length \ \text{Width} \ \text{curvedTree}>
\]

\textbf{Figure 2.58: Generated GDS fractal examples shapes.}
2.7.4 Function Plot

Functional plot method allows functional representation in \((x,y)\) and \((r,\theta)\) coordinate systems. Generated curves are defined by a function, translation \((x_L, y_L)\) and upper limits \((x_U, y_U)\), number of segments \((N)\), width \((W)\), cap and join parameters. Cap is represented by the following integer values \(BUTT = 0\), \(ROUND = 1\), or \(SQUARE = 2\). Join is represented by the following integer values \(MITER = 0\), \(ROUND = 1\), \(BEVEL = 2\). In order to avoid exceeding GDS shape point limits, each function is fractured horizontally and vertically about the centroid.

\[
\langle \{ y(x) \} \rangle \ x\ y\ x_L\ x_U\ N\ W\ CAP\ JOIN\ \theta_{(x,y)} \ functionXY >
\]

\[
\langle \{ r(t) \} \rangle \ x\ y\ \theta_L\ \theta_U\ N\ W\ CAP\ JOIN\ \theta_{(x,y)} \ functionRT >
\]

Figure 2.59: Function plots (a) \(y(x)\) and (b) \(r(\theta)\).

Functional plots in figures 2.59(a) and 2.59b are respectively rotated by 180° and 22.5° about \((10, -10)\). Both curves were constructed using \(N = 1000\), a width of 0.22 \(\mu m\), \(CAP = ROUND\) and \(JOIN = MITER\). Functional representation for the two curves is correspondingly given by

\[
y(x) = 14\sin\left(\frac{x}{2}\right)e^{-\frac{x}{44}} \\
r(\theta) = 11\sin(4\theta)
\]

The following scripts (see example file scriptingFunctionPlot.cnst) to generate functional plots in Figure 2.59:

\[
\langle \{ 14\cdot Math.sin(x/2)\cdot Math.exp(-x/44) \} \rangle \ 10\ -10\ 0\ 88\ 1000\ 0.22\ 1\ 0\ 180\ functionXY>
\]

\[
\langle \{ 11\cdot Math.sin(4\cdot t) \} \rangle \ 10\ -10\ 6.28\ 1000\ 0.22\ 1\ 0\ 22.5\ functionRT>
\]

As seen above, in polar coordinates \((r,\theta)\), \(\theta\) parameter is represented as a variable \(t\). Mathematical functions are represented in terms of the Java
Math class methods. For instance \( \sin(x) \) is represented as \( \text{Math.sin}(x) \). Several methods from the Java Math class are shown below. The entire Java Math class method summary documentation, including argument limits, is available online.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Math.sin}(x)</td>
<td>\sin(x)</td>
</tr>
<tr>
<td>\text{Math.cos}(x)</td>
<td>\cos(x)</td>
</tr>
<tr>
<td>\text{Math.tan}(x)</td>
<td>\tan(x)</td>
</tr>
<tr>
<td>\text{Math.sinh}(x)</td>
<td>\sinh(x)</td>
</tr>
<tr>
<td>\text{Math.cosh}(x)</td>
<td>\cosh(x)</td>
</tr>
<tr>
<td>\text{Math.tanh}(x)</td>
<td>\tanh(x)</td>
</tr>
<tr>
<td>\text{Math.exp}(x)</td>
<td>(e^x)</td>
</tr>
<tr>
<td>\text{Math.expm}(x)</td>
<td>(e^x - 1)</td>
</tr>
<tr>
<td>\text{Math.log}(x)</td>
<td>(\ln(x))</td>
</tr>
<tr>
<td>\text{Math.log10}(x)</td>
<td>(\log_{10}(x))</td>
</tr>
<tr>
<td>\text{Math.sqrt}(x)</td>
<td>(\sqrt{x})</td>
</tr>
<tr>
<td>\text{Math.pow}(x,n)</td>
<td>(x^n)</td>
</tr>
<tr>
<td>\text{Math.PI}</td>
<td>(\pi)</td>
</tr>
<tr>
<td>\text{Math.random}</td>
<td>random number in ([0,1))</td>
</tr>
</tbody>
</table>
2.7.5 Grayscale

2.7.5.1 Polygons

This section offers a variety of means for constructing grayscale objects from segmented polygons. GDS layers are assigned to each of the sections with layer numbers ranging from \(n-1\) (outer) to 0 (inner). Figures 2.60a, b and c show evenly distributed grayscale structure are centered at \((x,y)\), with \(n\) number of segmentations, the outermost dimensions defined by \(d_x, d_y, r_x\), and \(r_y\), and \(\theta_{(x,y)}\) rotation about the origin. Rectangular structures (figure 2.60a) are equivalent to 45 degree rotated \(n\)-gon structures where the number of sides \(N_s = 4\) (figure 2.60c).

\[
\begin{align*}
\text{grayER} & : x \ y \ d_x \ d_y \ n \ \theta_{(x,y)} \\
\text{grayENgon} & : x \ y \ r_x \ r_y \ n \ N_s \ \theta_{(x,y)}
\end{align*}
\]

Figure 2.60: Evenly distributed grayscale segments for (a) rectangles and \(n\)-gons with (b) large number of vertices \((N_s)\) and (c) \(N_s = 4, r_x = r_y\).
User defined grayscale segments are shown in figures 2.61a, b and c. The respective constructors are defined by \((d_{xi}, d_{yi})\) (or \((r_{xi}, r_{yi})\)) coordinate pairs. The outermost segment is defined by \((dx_1, dy_1)\) (or \((rx_1, ry_1)\)). The constructors further assume that \(dx_1 > dx_2 > dx_3 \cdots > dx_n\) and \(dy_1 > dy_2 > dy_3 \cdots > dy_n\). Similarly, for n-gon segments \(rx_1 > rx_2 > rx_3 \cdots > rx_n\) and \(ry_1 > ry_2 > ry_3 \cdots > ry_n\).

\[
\begin{align*}
\text{grayUDR} & : \quad x \ y \ d_{x_1} \ d_{y_1} \ d_{x_2} \ d_{y_2} \ \cdots \ d_{x_n} \ d_{y_n} \ \theta_{(x,y)} \\
\text{grayUDNgon} & : \quad x \ y \ r_{x_1} \ r_{y_1} \ r_{x_2} \ r_{y_2} \ \cdots \ r_{x_n} \ r_{y_n} \ N_s \ \theta_{(x,y)}
\end{align*}
\]

\(N_s\) \(r_{xi} = r_{yi}\).

**Figure 2.61:** User defined distribution of grayscale segments for (a) rectangles and n-gons with (b) large number of vertices \((N_s)\) and (c) \(N_s = 4, \ r_{x_1} = r_{y_1}\).
2.7.5.2 Ramp

greyERamp constructor allows ramping with either \( UP = 1 \), where the \( d_x \) is subdivided into \( n \) segments with GDS layer numbers range from 0 to \( n - 1 \). Alternatively, with \( UP = 0 \), the GDS layer numbers range from \( n - 1 \) to 0. greyERamp2 constructor forms a double ramp (up then down) with evenly divided segments where GDS layers range from

\[
L = \begin{cases} 
L^n_0 \ldots L_0 \ldots L^{n-1}_0 & n \text{ is even} \\
L^{n-1}_0 \ldots L_0 \ldots L^{n}_0 & n \text{ is odd} 
\end{cases} \tag{2.8}
\]

\[ x \quad y \quad d_x \quad d_y \quad n \quad UP \quad \theta_{(x,y)} \text{ greyERamp} \]

\[ x \quad y \quad d_x \quad d_y \quad n \quad \theta_{(x,y)} \text{ greyERamp2} \]

Figure 2.62: Evenly distributed grayscale ramp segments using greyERamp with UP values of (a) 1 and (b) 0. (c) Grayscale ramp up/down segments using the greyERamp2 constructor.
Similarly, user defined ramp segments are defined using the below constructors. Number of constructed shapes is equal to \( n - 1 \), where \( n \) is the number of defined segment positions. Grayscale segments undergo a translation by \((x_t, y_t)\) and rotation \( \theta(x_t, y_t) \). Constructors assume \( x_1 < x_2 < x_3 \cdots < d_{x_n}, n > 1 \) with GDS layers for \textit{grayUDRamp2} defined:

\[
L = \begin{cases} 
L_{\frac{n-1}{2}} \cdots L_0 \cdots L_{\frac{n-1}{2}} & n \text{ is even} \\
L_{\frac{n+1}{2}} \cdots L_0 \cdots L_{\frac{n+1}{2}} & n \text{ is odd and } n > 3 \\
L_0 \ L_1 & n = 3 \\
L_0 & n = 2 
\end{cases}
\]  

\[ (2.9) \]

\[
x_t \ y_t \ x_1 \ x_2 \ \ldots \ \ x_n \ d_y \ \text{UP} \ \theta(x_t, y_t) \ \text{grayUDRamp}
\]

\[
x_t \ y_t \ x_1 \ x_2 \ \ldots \ \ x_n \ d_y \ \theta(x_t, y_t) \ \text{grayUDRamp2}
\]

\[ (a) \quad (b) \quad (c) \]

**Figure 2.63:** User defined grayscale ramp segments using \textit{grayURRamp} with UP values of (a) 1 and (b) 0. (c) Grayscale ramp up/down segments using the \textit{grayURRamp2} constructor with an even number of points \( n \).
2.7.5.3 Spiral Staircase

The spiral structure is constructed using overlapping arc segments cast to different GDS layers \( (L_i) \). The resulting structure is defined by the center position \((x, y)\), inner and outer radii \((r_i \text{ and } r_o)\), number of arc segments \((n)\), number of sides of each segment \((N_{sides})\) and the rotation angle \((\theta_{(x,y)})\).

\[
x \quad y \quad r_i \quad r_o \quad n \quad N_{sides} \quad \theta_{(x,y)} \quad \text{graySpiralStairOverlap}
\]

Figure 2.64: Grayscale spiral staircase with overlapping segments. (a) 3D projection and (b) 2D illustration showing various constructor parameters. (c) Scanning electron micrograph of a focused ion beam fabricated structure. Scale bar corresponds to 200 nm. (Courtesy of L. Ocola [82])
The following constructor creates a grayscale spiral staircase structure without overlapping arc segments.

```
x  y  r_i  r_o  n  N_{sides}  \theta_{(x,y)}  \text{graySpiralStair}
```

Here, each segment \((n_i)\) is represented by a distinct GDS layer.

Figure 2.65: Grayscale spiral staircase.
2.7.6 Interdigitated Electrodes

\[ x \ y \ w_1 \ w_2 \ l_1 \ l_2 \ l_3 \ N \ p \ b_H \ b_W \ \theta_{(x,y)} \ \text{intElec1} \]

\[ x \ y \ w_1 \ w_2 \ l_1 \ l_2 \ l_3 \ N \ p \ b_H \ b_W \ \theta_{(x,y)} \ \text{intElec2} \]

Figure 2.66: Interdigitated electrodes (a) intElec1 and (b) intElec2.
Figure 2.67: Interdigitated electrodes (a) intElec3 and (b) intElec4.
Figure 2.68: Interdigitated electrodes intElec5.
2.7.7 Junctions

2.7.7.1 T Junction

\[ x \ y \ w_1 \ w_2 \ L_1 \ L_2 \ r \ N_{sides} \ \theta_{(x,y)} \ \text{tJunction} \]

![Diagram of T junction with and without circular ports](image)

Figure 2.69: T junction (a) with and (b) without \( r = 0 \) circular ports.
2.7.7.2 H Junction

\[
\begin{aligned}
&x \quad y \quad w_1 \quad w_2 \quad L_1 \quad L_2 \quad r \quad N_{sides} \quad \theta_{(x,y)} \\
&\text{hJunction}
\end{aligned}
\]

(a)

(b)

Figure 2.70: H junction (a) with and (b) without \( r = 0 \) circular ports.
2.7.7.3 Arrow Junction

\[ x \ y \ w_1 \ w_2 \ L_1 \ L_2 \ r \ N_{sides} \ \theta \ \theta_{(x,y)} \text{ arrowJunction} \]

(a)

(b)

Figure 2.71: Arrow junction (a) with and (b) without \( r = 0 \) circular ports.
2.7.8 Meander Channel

The method constructs a meandering channel with rectangular ports. These structures find potential use in fluidic and gas delivery system applications. Four available constructors represent various wavelike structures including sine, square, ramp and triangle. Rectangular ports, residing at each channel end, are defined by the length \( L_1 \) and \( L_2 \) and height \( H_1 \) and \( H_2 \) parameters. Connections to the reservoirs are made through segments \( a \) and \( b \) of width \( W \). Wave structures initiate at \( x = a \) and are of length \( b \). Number of wave periods is defined by \( N \). The sine wave structure is constructed from a number of segments defined by the parameter \( N_{\text{segments}} \).

\[ \begin{align*}
<x, y, L_1, H_1, L_2, H_2, W, A, N, N_{\text{segments}}, a, b, c, \theta_{(x,y)} &> \text{ meanderSin} > \\
<x, y, L_1, H_1, L_2, H_2, W, A, N &> a, b, c, \theta_{(x,y)} \text{ meanderSqr} > \\
<x, y, L_1, H_1, L_2, H_2, W, A, N &> a, b, c, \theta_{(x,y)} \text{ meanderRamp} > \\
<x, y, L_1, H_1, L_2, H_2, W, A, N &> a, b, c, \theta_{(x,y)} \text{ meanderTri} > 
\end{align*} \]

Figure 2.72 illustrates the four meander channel configurations with respective parameters. As shown in Figure\ref{meanderPortsExample}, rectangular reservoirs at each channel end can be eliminated by setting respective length and height parameters equal to 0, i.e. \( L_1 = H_1 = 0 \) and/or \( L_2 = H_2 = 0 \).
Figure 2.72: Example shapes illustrating various parameters from the (a) meanderSin, (b) meanderSqr, (c) meanderRamp and (d) meanderTri constructors.
Figure 2.73: Meander channel example illustrating various rectangular reservoir configurations. Structures without (a) right reservoir using $L_2 = H_2 = 0$, (b) left reservoir using $L_1 = H_1 = 0$ and (c) reservoirs using $L_1 = H_1 = L_2 = H_2 = 0$. 
2.7.9 Points To Shape

The method creates a polygon from user defined $(x, y)$ point pairs. Last point $(x_n, y_n)$ connects the first $(x_1, y_1)$ to close the shape.

![Diagram of points forming a polygon]

**Figure 2.74:** Example of the `points2shape` constructor illustrating a closed shape constructed from points 5 points, $(x_1, y_1) \ldots (x_5, y_5)$.

**Figure 2.75:** `points2shape` example illustrating a GDS rendered polygon.
2.7.10 Polygon Along a Path

The method creates a polygon of a specified width ($W$) along a path defined by $(x, y)$ point pairs. Cap is represented by the following integer values $BUTT = 0$, $ROUND = 1$, or $SQUARE = 2$. Join is represented by the following integer values $MITER = 0$, $ROUND = 1$, $BEVEL = 2$.

![Diagram of polygon along a path]

Figure 2.76: Example of the polypath constructor illustrating a rendered shape constructed from points 5 points, $(x_1, y_1)\ldots(x_5, y_5)$.

Polypath example illustrating a GDS polygon along a Gaussian path.

$\begin{array}{ccccccccc}
x_1 & y_1 & x_2 & y_2 & \cdots & x_n & y_n & W & Cap & Join & polypath
\end{array}$

Figure 2.77: polypath example illustrating a GDS polygon along a Gaussian path.
2.7.11 Random Polygons

The below constructor creates randomly placed polygons into a user defined area. Single polygon of radius \( r \) with \( N_s \) sides is created and cast into a GDS struct with the name \texttt{uniqueStructName}. The \( N_e \) number of shapes are then instantiated into an area of width \( W \) and height \( H \). Separation parameter \( S \) defines the minimum separation between the outer radial perimeter of the objects. If the randomly generated coordinate violates the minimum separation distance, the module will keep generating random coordinates until the maximum number of failures (defined by the parameter iteration parameters \( I \)) is reached. The lower left corner of the random array is positioned at \((x,y)\).

Random rotation is enabled if \( R_R \) is set to 1, each shape will be randomly rotated within the current GDS structure. Any other numeric value or \( R_R \) will leave the shape element unchanged. Enabling the \( R_R \) option for features with many sides is not particularly useful, however, with \( S = 3 \) and \( R_R = 1 \) produces randomly oriented triangles (see Figure 2.78).

\[
<\texttt{uniqueStructName} \ x \ y \ W \ H \ r \ N_s \ S \ N_e \ I \ R_R \ \texttt{randomPolygons}>
\]

Figure 2.78: View of a generated GDS file using the Random Polygons module. The area was \( 40\mu m \times 40\mu m \), number of elements and iterations were set to 400, with an element radius and separation of \( 1\mu m \) and \( 2\mu m \) respectively. Generated shapes had (a) 44 sides and (b) 3 sides with random rotation enabled.
2.7.12 Random Ellipses and Vectorized Ellipses

Similar to the random polygon example, the following constructors create random ellipses (randomEllipses) and vectorized ellipses (randomEllipsesV). shapeReso defines the rendering resolution of randomEllipsesV.

\[
\begin{align*}
\text{randomEllipses} & \quad \langle \text{uniqueStructName} \ x \ y \ W \ H \ rx \ ry \ N_s \ S \ N_e \ I \ RR \rangle \\
\text{randomEllipsesV} & \quad \langle \text{uniqueStructName} \ x \ y \ W \ H \ rx \ ry \ S \ N_e \ I \ RR \rangle
\end{align*}
\]

Figure 2.79: Random ellipse example showing separation (S) between adjacent circular boundaries defined by the radius ry.

Figure 2.80: GDS rendered random ellipses with random rotation enabled (RR = 1).
2.7.13 Resolution Test Pattern

Radial pattern below is useful for characterizing the stigmation of a lithography tool. Using the `resoPattern` constructor, the structure is defined by the center origin \((x, y)\), radius \(r\) and width \(w\). In this scenario, to ensure pattern symmetry, \(360/w = i\), where \(i = 2, 4, \ldots, n\) is an even positive integer. For even distribution of segments across the \(2\pi r\) circumference, \(i\) can be any positive integer greater than zero. The second constructor (`resoPatternPi`) creates a similar figure where the end-segment is an integer of \(\pi\) width.

\[
\begin{align*}
  x & \quad y & \quad r & \quad w & \quad \theta_{(x,y)} & \quad \text{resoPattern} \\
  x & \quad y & \quad r & \quad n & \quad \theta_{(x,y)} & \quad \text{resoPatternPi}
\end{align*}
\]

\[(a)\]

\[(b)\]

Figure 2.81: Resolution patterns using (a) `resoPattern` and (b) `resoPatternPi` constructors.
Resolution pattern formed using two rows of rectangles of width $W$, height $H$, pitch $2W$ and $N$ elements in each row. A text label is placed directly above the array of lines indicating the value of $W$. The font resolution is defined using the global `shapeReso` parameter. The font height is equal to the value of $H$ (text label in Figure 2.82 is not shown to scale).

$x \ y \ W \ H \ N \ \theta_{(x,y)} \ \text{resoPatternRS}$

The following constructor creates arrays of rectangular shaped resolution patterns using a start and end width values ($W_s$ and $W_e$), with a $\delta$ incremental variation. The space between the arrays is defined by the parameter $S$ in micrometer units.

$x \ y \ W_s \ W_e \ \delta \ H \ N \ S \ \theta_{(x,y)} \ \text{resoPatternRSA}$

![Figure 2.82: Resolution pattern of rectangular lines using (a) resoPatternRS and (b) resoPatternRSA constructors.](image-url)
The following resolution pattern is formed using L-shapes of width \( W \), height \( H \), pitch \( 2W \) and \( N \) number of shapes. The maximum number of generated shapes is defined when \( W = H \). Text label definition is identical to one described for the \texttt{resoPatternRS} constructor (previous example).

\[
x \ y \ W \ H \ N \ \theta_{(x,y)} \quad \texttt{resoPatternLS}
\]

An arrayed version of the above constructor is defined by a start and end width values \((W_s, W_e)\), with a \( \delta \) incremental variation. The space between the arrays is defined by the parameter \( S \) in micrometer units.

\[
x \ y \ W_s \ W_e \ \delta \ H \ N \ S \ \theta_{(x,y)} \quad \texttt{resoPatternLSA}
\]

![Diagram](image)

Figure 2.83: Resolution pattern of rectangular lines using (a) \texttt{resoPatternLS} and (b) \texttt{resoPatternLSA} constructors.
2.7.14 Spirals

Spiral objects are characterized by uniform, converging and diverging spacing between subsequent turns. Constant spacing represents an Archimedes spiral where \( r = m \theta \), where \( m = (\text{separation} + \text{width})/(2\pi) \) (Figure 2.84a). Separation parameter defines the pitch between subsequent spiral turns. Fermat’s spiral, defined as \( r = \sqrt{a^2 + \theta^2} \), is shown in Figure 2.84b. The logarithmic spiral, defined as \( r = a e^{b \theta} \), is shown in Figure 2.84c. Center of each spiral is defined by the point \((x, y)\).

![Spiral Examples](image)

**Figure 2.84**: GDS rendered spiral examples. (a) Archimedes (b) Fermat and (c) Logarithmic spiral.

The following scripts were used to generate the spirals centered around the origin \((x = 0, y = 0)\) in Figure 2.84:

- Archimedes:
  - \( x y \) \( \text{Width} \) \( N_{\text{turns}} \) \( \text{Separation} \) \( \text{increment} \) \( \text{spiralArch} \)
  - 0 0 1.0 5 2.0 0.010 spiralArch
  - 0 0 2.5 7 8 0.010 spiralFermat
  - 0 0 4.4 4 8.0 0.1 0.010 spiralLog
2.7.15 Spiral - Rectangular

Below constructor creates a rectangular spiral of $N_{\text{turns}}$, of width $w$, start length $L_s$ and pitch $p$ between turns.

\[ x \ y \ w \ L_s \ p \ N_{\text{turns}} \ \theta_{(x,y)} \ \text{spiralRect} \]

Figure 2.85: Rectangular spiral.
2.8 \textbf{Alignment and Reticle Elements}

2.8.1 Alignment Marks - Predefined

The following section includes a variety of predefined, multilevel alignment mark strategies for lithographic applications. Within these methods, alignment mark size and shape values are predefined. The central dark circular aperture within each of the structures defines the \((x, y)\) position. The \(\theta_{(x,y)}\) defines the rotation (in degrees) about \((x, y)\).

The following are constructors for two- and three-level alignment marks shown in figures 2.86 and 2.87. \(L_a, L_b\) and \(L_c\) are integer values \((0 - 255)\) representing GDS layer numbers. Pattern extents for \texttt{alignFFFFB1} and \texttt{alignFFFFB2} alignment patterns in figure 2.86 range from \((-80\mu m, -80\mu m)\) to \((80\mu m, 80\mu m)\). For the three-level alignment mark system (\texttt{align3Level}), pattern extents range from \((-1000\mu m, -1000\mu m)\) to \((1000\mu m, 1000\mu m)\).

\[
\begin{align*}
&<x\ y\ L_a\ L_b\ \theta_{(x,y)}\ \texttt{alignFFFFB1}> \\
&<x\ y\ L_a\ L_b\ \theta_{(x,y)}\ \texttt{alignFFFFB2}> \\
&<x\ y\ L_a\ L_b\ L_c\ \theta_{(x,y)}\ \texttt{align3Level}>
\end{align*}
\]

The following defines a constructor for a two level alignment system with verniers and options for tone reversal of layers within a specified region (figure 2.88). As defined above, \(L_a\) and \(L_b\) are integer values defining the two rendered GDS layer numbers. Parameter \texttt{vernReso} defines the resolution between the two verniers in units of \(\mu m\). Layers \(L_a\) and/or \(L_b\) are tone-reversed within an area defined by the inversion length \((I_L)\) and width \((I_W)\) if their respective boolean flags \(INV_a\) and \(INV_b\) are activated. The boolean inversion parameters are activated using any integer value other than 0. Pattern extents range from \((-900\mu m, -900\mu m)\) to \((900\mu m, 900\mu m)\).

\[
\begin{align*}
&<x\ y\ L_a\ L_b\ \texttt{vernReso}\ INV_a\ INV_b\ I_L\ I_W\ \theta_{(x,y)}\ \texttt{alignVern}> \\
&<x\ y\ L_a\ L_b\ \texttt{vernReso}\ \theta_{(x,y)}\ \texttt{alignVernLb1}> \\
&<x\ y\ L_a\ L_b\ \texttt{vernReso}\ \theta_{(x,y)}\ \texttt{alignVernLb2}>
\end{align*}
\]

Two-level alignment strategies defined in figure 2.89 contains verniers and GDS layer number labels. As defined directly above, \(L_a\) and \(L_b\) are GDS layer numbers, and \texttt{vernReso} parameter defines the resolution between the two verniers. \texttt{alignVernLb1} pattern extents in range from \((-1000\mu m, -1000\mu m)\) to \((1000\mu m, 1000\mu m)\), whereas \texttt{alignVernLb2} pattern extents in range from \((-230\mu m, -230\mu m)\) to \((117\mu m, 155\mu m)\).

\[
\begin{align*}
&<x\ y\ L_a\ L_b\ \texttt{vernReso}\ \theta_{(x,y)}\ \texttt{alignVernLb1}> \\
&<x\ y\ L_a\ L_b\ \texttt{vernReso}\ \theta_{(x,y)}\ \texttt{alignVernLb2}>
\end{align*}
\]
Figure 2.86: Two level lithography alignment mark strategies constructed using (a) alignFFFB1 and (b) alignFFFB2. Two layers $L_a$, and $L_b$ are shown separately as blue (left) and orange (right). These methods could be used for front-to-front or front-to-back alignment.
Figure 2.87: Three-level, front-side, lithography alignment mark strategy constructed using align3Level with layers (a) $L_a$, (b) $L_b$ and (c) $L_c$.

Figure 2.88: Two-level alignment mark strategy with $x$ and $y$ verniers constructed using alignVern. Blue and orange layers are $L_a$ and $L_b$ respectively. Length ($L$) and width ($W$) parameters define the region extent used for tone reversal of the specified alignment layers. This method is suitable for conventional overlay lithography applications.
Figure 2.89: Two-level alignment mark strategies with $x$ and $y$ verniers, and layer labels constructed using (a) `alignVernLb1` and (b) `alignVernLb2`. Blue and orange layers are $L_a$ and $L_b$ respectively. This method is suitable for conventional overlay lithography applications.
2.8.2 Alignment Marks - Custom

The following section includes 4 types of cross-based marks for multilevel lithographic alignment. Unlike section 2.8.1 with predefined marks, the following mark definitions are fully customizable by user defined parameters. Figure 2.90 shows a small circle at the center of each shape denoting the translation point \((x, y)\). Each shape undergoes a rotation by \(\theta_{(x,y)}\) about the center point \((x, y)\). The dotted box of length \(I_L\) and width \(I_W\) denotes the boundary within which layer inversion takes place. Boolean tone reversal (inversion) is activated by the parameter \(INV\). A value \(INV = 0\) signifies that tone reversal within the specified boundary will not take place. Any other integer value for \(INV\) will invert the tone of the mark. Layers are set by the usual layer command (see section 2.2.4).

Figure 2.90a shows a simple cross defined by a width \((W)\), a horizontal and vertical length segments \((L_1 \text{ and } L_2)\).

\[
<x \ y \ L_1 \ W \ L_2 \ INV \ I_L \ I_W \ \theta_{(x,y)} \ \text{alignCustC1}>
\]

Figure 2.90b shows a cross of width \((W)\), length \((L)\), paddle length \((P_L)\) and width \((P_W)\).

\[
<x \ y \ L \ W \ P_L \ P_W \ INV \ I_L \ I_W \ \theta_{(x,y)} \ \text{alignCustC2}>
\]

Figure 2.90c shows a cross of width \((W)\), length \((L)\), paddle length \((P_L)\) and width \((P_W)\) offset by a length \((L_X)\) from the edge.

\[
<x \ y \ L_1 \ W \ P_L \ P_W \ L_X \ INV \ I_L \ I_W \ \theta_{(x,y)} \ \text{alignCustC3}>
\]

Figure 2.90d shows a cross defined by a width \((W)\), a horizontal and vertical length segments \((L_1 \text{ and } L_2)\), and a rectangle of length \((B_L)\) and width \((B_W)\) at a distance \((d)\) from the cross edge.

\[
<x \ y \ L_1 \ W \ L_2 \ d \ B_L \ B_W \ INV \ I_L \ I_W \ \theta_{(x,y)} \ \text{alignCustC4}>
\]
Figure 2.90: Custom alignment crosses using (a) `alignCustC1`, (b) `alignCustC2`, (c) `alignCustC3`, (d) `alignCustC4` constructors.
2.8.3 Reticle Barcode and Label Frames

The following methods create reticle marks, barcodes and labels for the CNST ASMLPAS5500 i-line stepper and reticle labels for contact lithography tools (5 inch and 7 inch). Contact reticle frames contain a label (up to 22 characters) along with a NIST and CNST logos. In all cases MIRROR is a parameter that defines if the final structure will be mirrored. Any integer other than 0 will mirror the resulting reticle frame.

CNST ASML PAS5500 stepper frame with reticle marks, barcode, label and logo:

```xml
<{{labelText}} {{barCode}} MIRROR cnstASML>
```

Contact Frames:

```xml
<{{labelText}} MIRROR cnstContact5>
<{{labelText}} MIRROR cnstContact7>
```

Figure 2.91: CNST ASML PAS5500 reticle frame with MIRROR= 0.
2.8.4 Vernier

Figure 2.92 shows a vernier structure with a central element of length ($L$), width ($W$), and pitch ($p$). Dotted rectangle of length ($I_L$) and width ($I_W$) represent a boundary within which layer tone reversal takes place. With inversion parameters set to 0, i.e. $I_a = 0$ and/or $I_b = 0$, boolean inversion of layers $L_a$ and/or $L_b$ will not take place. Any other integer value will invert the pattern within the bounding box. Parameter $vernReso$ defines the resolution of the verniers.

$$<x \ y \ L_a \ L_b \ L \ W \ p \ vernReso \ s \ I_a \ I_b \ I_L \ I_W \ \theta_{(x,y)} \ alignCustVern>$$

Figure 2.92: Verniers between layers $L_a$ (blue) and $L_b$ (orange) created using the alignCustVern constructor.
2.8.5 Vernier With Labels

The method constructs verniers between two lithographic levels. Text resolution is set by `shapeReso`. Text font used is Serif. The verniers are centered at position \((x, y)\). Vernier resolution is in micrometers. The two layer parameters, \(LyrA\) and \(LyrB\) are GDS layer numbers (integers from 0 to 255), Parameter \(N_{ticks}\) represents the total number of vernier bars. Text labels are defined by the `fontSize` and string parameters \(LblA\) and \(LblB\). These strings cannot have white characters (spaces/tabs) and are generally short layer descriptions. The width, length and pitch parameters define the vernier bars.

**Figure 2.93:** Example of the verniers constructor illustrating parameters between two layers.

```xml
<x y LyrA LyrB vernierReso N_{ticks} LblA LblB fontSize W L Pitch verniers>
```

**Figure 2.94:** GDS rendered output of the verniers example between two lithographic levels.
2.8.6 Arrows

This method constructs arrow structures that are useful in a variety of lithographic applications including labeling features and as an aid in finding features of interest, for instance, arrows pointing towards alignment marks or small, isolated devices. Each constructor is defined by parameters in the below table. An isolated arrow head is used to construct the arrow element. Constructor arrowArray defines a linear array of N arrow features.

\[<x \ y \ W_a \ L_a \ W \ \theta_{(x,y)} \ \text{arrowHead}>\]
\[<x \ y \ W_a \ L_a \ W \ L \ \theta_{(x,y)} \ \text{arrow}>\]
\[<x \ y \ W_a \ L_a \ W \ L \ N \ \theta_{(x,y)} \ \text{arrowArray}>\]

![Diagram of arrow constructs](image)

**Figure 2.95:** Example shapes illustrating various parameters from the (a) arrowHead, (b) arrow and (c) arrowArray constructors.

**Figure 2.96:** Rendered GDS file example with arrowArray structures.
This section contains a variety of shapes commonly encountered nanophotonic applications. Many of these shapes (tapers, S-bends, Y-bends, etc) could be used for micro- and nano-fluidics, MEMS/NEMS and other fields of applied sciences. Each element has several constructor definitions, allowing the user to create structures using either positive or negative tone resists. Additionally, constructors allow for addition of endcaps at the structural termination points. This in turn alleviates stresses at sharp field crowding corners, thereby preventing undesired formation of stress cracks. Figure 2.97 shows an example of 600 nm patterned ZEP layer on top of 700 nm stoichiometric silicon nitride deposited using low pressure chemical vapor deposition. In this case, cracks initiated at the sharp edges within the nitride layer during the reactive ion etch step (figure 2.97a, b). The dilemma was circumvented by corner rounding (figure 2.97c, d).

**Figure 2.97:** Silicon nitride waveguides with square ((a) and (b)) and rounded corners ((c) and (d)). Waveguides with square corners show cracks initiating at the waveguide edges. Waveguides with rounded corners alleviate stresses at sharp corners, thereby preventing cracks formation. Scale bar in (a) and (c) represents 250\(\mu\)m and in (b) and (d) represents 20\(\mu\)m.
2.9.1 Waveguides

2.9.1.1 Waveguide

This method is useful when creating waveguides with a negative tone resist. Rectangular shape characterized by start point \((x_1, y_1)\), end point \((x_2, y_2)\), width \((W)\) and rotation about point \((x_1, y_1)\). The waveguide is defined in the horizontal direction, condition i.e. \(x_1 = x_2\) must be satisfied. For vertical waveguides, choose \(\theta_{(x_1, y_1)} = 90\). Waveguide endcaps are defined by parameters \(EC_{left}\) and \(EC_{right}\). Endcap values of 0 and 1 correspond to slot waveguides without and with endcaps.

\[
\langle x_1 \quad y_1 \quad x_2 \quad y_2 \quad W \quad \theta_{(x_1, y_1)} \quad EC_{left} \quad EC_{right} \quad \text{waveguide}\rangle
\]

\(a\)

\(b\)

\(c\)

Figure 2.98: Waveguide example using the waveguide constructor. The method is beneficial when creating waveguides from a negative tone resist. Waveguides examples (a) without endcaps \((E_{left} = E_{right} = 0)\), (b) with left end cap \((E_{left} = 1, E_{right} = 0)\), (c) with right endcap \((E_{left} = 0, E_{right} = 1)\).
2.9.1.2 Waveguide Slot

This method is useful when creating slot waveguides with a negative tone resist. Positive-tone resist exposure creates a waveguide of width $W_{\text{slot}}$. The shape characterized by start point $(x_1, y_1)$, end point $(x_2, y_2)$, waveguide width ($W$), slot width ($W_{\text{slot}}$) and rotation about point $(x_1, y_1)$. The waveguide is defined in the horizontal direction, condition i.e. $x_1 = x_2$ must be satisfied. For vertical waveguides, choose $\theta(x_1, y_1) = 90$. Waveguide endcaps are defined by parameters $EC_{\text{left}}$ and $EC_{\text{right}}$. Endcap values of 0 and 1 correspond to slot waveguides without and with endcaps.

$<x_1 \ y_1 \ x_2 \ y_2 \ W \ W_{\text{slot}} \ \theta(x_1,y_1) \ EC_{\text{left}} \ EC_{\text{right}} \ \text{waveguideSlot}>$

![Diagram of waveguideSlot constructor](image)

(a)

![Diagram of waveguideSlot constructor](image)

(b)

Figure 2.99: Slot waveguide example using the waveguideSlot constructor. The method is beneficial when creating waveguides from a negative tone resist. Positive-tone resist exposure creates a waveguide of width $W_{\text{slot}}$. Slot waveguide examples (a) without endcaps ($E_{\text{left}} = E_{\text{right}} = 0$), (b) with right end cap ($E_{\text{left}} = 0$, $E_{\text{right}} = 1$).
2.9.1.3 Waveguide Inverse

This method is useful when creating waveguides with a positive tone resist. The shape characterized by start point \((x_1, y_1)\), end point \((x_2, y_2)\), waveguide width \((W)\), exposed sleeve width \((W_e)\) and rotation about point \((x_1, y_1)\). The waveguide is defined in the horizontal direction, condition i.e. \(y_1 = y_2\) must be satisfied. For vertical waveguides, choose \(\theta(x_1,y_1) = 90\). Waveguide endcaps are defined by parameters \(EC_{left}\) and \(EC_{right}\). Endcap values of 0 and 1 correspond to slot waveguides without and with endcaps.

\[
\langle x_1 \ y_1 \ x_2 \ y_2 \ W \ W_e \ \theta_{(x_1,y_1)} \ EC_{left} \ EC_{right} \ \text{waveguidelInv}\rangle
\]

Figure 2.100: Waveguide example using the waveguidelInv constructor. The method is beneficial when creating waveguides from a positive tone resist. Inverse waveguide examples (a) without endcaps \((E_{left} = E_{right} = 0)\), (b) with right end cap \((E_{left} = 0, E_{right} = 1)\).
2.9.1.4 Waveguide Inverse Slot

This method is useful when creating slot waveguides with a positive tone resist. The shape characterized by start point \((x_1, y_1)\), end point \((x_2, y_2)\), waveguide width \((W)\), slot width \((W_s)\), exposed sleeve width \((W_e)\) and rotation about point \((x_1, y_1)\). The waveguide is defined either in the horizontal or vertical direction, condition i.e. \(x_1 = x_2\) must be satisfied.

\[
\begin{align*}
<x_1 & y_1 x_2 y_2 W W_s W_e \theta(x_1,y_1) EC_{left} EC_{right} \text{ waveguidInvSlot}>
\end{align*}
\]

![Waveguide Inverse Slot Diagram](image)

**Figure 2.101:** Inverse slot waveguide example using the `waveguidInvSlot` constructor. The method is beneficial when creating waveguides from a positive tone resist. Inverse slot waveguide examples (a) without endcaps \((E_{left} = E_{right} = 0)\), (b) with right end cap \((E_{left} = 0, E_{right} = 1)\).
2.9.1.5 Waveguide Expander

The waveguide expander is defined by width $w$, length $L$, length division $\Delta L$, TE mode angle $\theta$, beam waist $w_0$, wavelength $\lambda$, base height $b_H$ and two dimensionless parameters $a$ and $b$. Wavelength $\lambda$ is represented in nanometers. All other parameters ($L, \Delta L, w, w_0, b_H$) are in micrometers. The variable gap is defined by

$$g(x) = \frac{1}{b} \ln \left( \frac{2 \lambda \pi^{1.5}}{aw} \exp \left( \frac{-x^2}{w^2} \right) \right) \left( \frac{1 - erf \left( \frac{a}{w} \right)}{1 - erf \left( \frac{b}{w} \right)} \right) \right)$$

(2.10)

where

$$w = \frac{w_0}{\sin \theta}$$

(2.11)

$x\ y\ w\ L\ \Delta L\ a\ b\ \theta\ w_0\ \lambda\ b_H\ \theta_{(x,y)}$ $\text{wgExpander}$

Figure 2.102: Waveguide expander
2.9.2 Tapers

2.9.2.1 Linear Taper

The linear waveguide taper method is beneficial with negative tone resists. Trapezoidal shape characterized by start point \((x_1, y_1)\), end point \((x_2, y_2)\), width at start \((W_1)\) and end \((W_2)\) points, and rotation about point \((x_1, y_1)\). The linear taper shape is defined either in the horizontal or vertical direction, condition i.e. \(x_1 = x_2\) or \(y_1 = y_2\) must be satisfied.

\[
\begin{bmatrix}
  x_1 \\
y_1 \\
x_2 \\
y_2 \\
W_1 \\
W_2 \\
\theta_{(x_1,y_1)}
\end{bmatrix}
\text{ linearTaper}
\]

Figure 2.103: Linear waveguide taper generated using the \texttt{linearTaper} constructor. Structure is useful with negative tone resists.
2.9.2.2 Linear Taper Slot

The slot linear waveguide taper is beneficial with negative tone resists. With positive tone resists, this structure would result in a tapered waveguide structure. Trapezoidal shape characterized by start point \((x_1, y_1)\), end point \((x_2, y_2)\), width at start \((W_1)\) and end \((W_2)\) points, slot width at start \((W_{s1})\) and end \((W_{s2})\) points, and rotation about point \((x_1, y_1)\). The linear taper shape is defined either in the horizontal or vertical direction, condition i.e. \(x_1 = x_2\) or \(y_1 = y_2\) must be satisfied.

\[
\langle x_1 \ y_1 \ x_2 \ y_2 \ W_1 \ W_2 \ W_{s1} \ W_{s2} \ \theta_{(x_1,y_1)} \text{ linearTaperSlot} >
\]

Figure 2.104: Linear taper slot waveguide structure created using the linearTaperSlot constructor. Structure is useful with negative tone resists. Positive tone resists generate a tapered waveguide structure of start and end with of \(W_{s1}\) and \(W_{s3}\) respectively.
2.9.2.3 Linear Taper Inverse Slot

The inverse slot linear waveguide taper is beneficial with negative tone resists. With positive tone resists, this structure would result in a tapered slot waveguide structure. Trapezoidal shape characterized by start point \((x_1, y_1)\), end point \((x_2, y_2)\), width at start \((W_1)\) and end \((W_2)\) points, slot width at start \((W_{s1})\) and end \((W_{s2})\) points, gap \((g)\), and rotation about point \((x_1, y_1)\). The linear taper shape is defined either in the horizontal or vertical direction, condition i.e. \(x_1 = x_2\) or \(y_1 = y_2\) must be satisfied.

\[
<x_1 \ y_1 \ x_2 \ y_2 \ W_1 \ W_2 \ g \ W_{s1} \ W_{s2} \ \theta_{(x_1,y_1)} \ \text{linearTaperInvSlot}>
\]

Figure 2.105: Example of the \text{linearTaper} constructor.
2.9.2.4 Exponential Taper

Tapered shape characterized by a width \( W_1 \) at start point \((x_1, y_1)\) exponentially increasing to \( W_2 \) at a length \( L \) from \((x_1, y_1)\), and rotation about point \((x_1, y_1)\). The exponential curve is constructed from \( N \) segments.

\[
<x_1 \ y_1 \ L \ W_1 \ W_2 \ N \ \theta_{(x_1,y_1)} \ exponentialTaper>
\]

![Figure 2.106: Example of the exponentialTaper constructor.](image_url)
2.9.2.5 Exponential Taper Inverse

Tapered shape characterized by a width \( W_1 \) at start point \((x_1, y_1)\) exponentially increasing to \( W_2 \) at a length \( L \) from \((x_1, y_1)\), exposed sleeve width \( W_e \) and rotation about point \((x_1, y_1)\). The exponential curve is constructed from \( N \) segments.

\[
< x_1 \quad y_1 \quad L \quad W_1 \quad W_2 \quad N \quad W_e \quad \theta(x_1, y_1) \quad \text{exponentialTaperInv} >
\]

Figure 2.107: Tapered exponential shape created using the exponentialTaperInv constructor.
2.9.2.6 Exponential Taper Inverse Slot

Tapered shape characterized by a width \( W_1 \) at start point \((x_1, y_1)\) exponentially increasing to \( W_2 \) at a length \( L \) from \((x_1, y_1)\), start and end slot widths \((W_{s1} \text{ and } W_{s2})\), exposed sleeve width \( W_e \) and rotation about point \((x_1, y_1)\). The exponential curve is constructed from \( N \) segments.

\[
<x_1 \ y_1 \ L \ W_1 \ W_2 \ N \ W_e \ W_{s1} \ W_{s2} \ \theta_{(x_1,y_1)} \ exponentialTaperInvSlot>
\]

Figure 2.108: Tapered exponential slot waveguide shape created using the exponentialTaperInvSlot constructor.
2.9.2.7 Custom Taper

Custom tapered shape characterized by user-defined \((x, y)\) coordinates. Points \((x_1, y_1)\) to \((x_n, y_n)\) are generated (top portion of the curve) and mirrored around \(y = 0\), closing the tapered shape that’s translated to \((T_x, T_y)\) and rotated about the translation point \(\theta(T_x, T_y)\).

\[
\begin{align*}
& x_1 \\ & y_1 \\ & x_2 \\ & y_2 \\ & \cdots \\ & x_n \\ & y_n \\ & T_x \\ & T_y \\ & \theta(T_x,T_y)
\end{align*}
\]

\textit{customTaper}

Figure 2.109: Two example GDS shapes of the \textit{customTaperExamples} constructor with highlighted user defined vertices.
2.9.3 Couplers

2.9.3.1 Directional Couplers

\(<x\ y\ L_1\ w_1\ w_{e1}\ L_2\ w_2\ w_{e2}\ g\ r\ N_{sides}\ \theta_{(x,y)}\ \text{directionalCoupler1}>\)

Figure 2.110: Directional coupler example using the \textit{directionalCoupler1} constructor.
Figure 2.111: Directional coupler example using the directionalCoupler2 constructor.
\[ (x, y, w, w_e, g, L_1, L_2, r, N_{\text{sides}}, \theta_{(x,y)}) \quad \text{directionalCoupler3} \]

Figure 2.112: Directional coupler example using the \texttt{directionalCoupler3} constructor.
Figure 2.113: Directional coupler example using the `directionalCoupler4` constructor.
2.9.3.2 S-Bend Taper

S-Bend taper is characterized by the start coordinate \((x, y)\), length \((L)\), height \((H)\), start \((W_{\text{start}})\) and end \((W_{\text{end}})\) waveguide widths, and rotation about point \((x, y)\). \texttt{shapeReso} parameter defines the rendering resolution of the S-bend (Bezier based) curve.

\[
\langle x \ y \ L \ H \ W_{\text{start}} \ W_{\text{end}} \ \theta_{(x,y)} \ \texttt{sBendTaper}\rangle
\]

![Figure 2.114: An example GDS shape of the sBendTaper constructor.](image)
2.9.3.3 S-Bend Funnel

S-Bend funnel is characterized by the start coordinate \((x, y)\), length \((L)\), height \((H)\), start \((W)\) and rotation about point \((x, y)\). `shapeReso` parameter defines the rendering resolution of the S-bend (Bezier based) curve.

\[
\begin{align*}
<x & \quad y & \quad L & \quad H & \quad W & \quad \theta_{(x,y)} & \quad \text{sBendFunnel}>
\end{align*}
\]

Figure 2.115: Schematic illustration showing various parameters from the `sBendFunnel` constructor.
2.9.3.4 S-Bend

S-Bends are Bezier curves characterized by two constructors. First consists of the start \((x_1, y_1)\) and end \((x_2, y_2)\) points. The second is characterized by the start point \((x_1, y_1)\), length \((L)\) and height \((H)\) parameters. Both constructors are characterized by a waveguide width \((W)\) and the rotation about the start point \((x_1, y_1)\). \texttt{shapeReso} parameter defines the rendering resolution of the S-bend (Bezier based) curve. Both \(L\) and \(H\) parameters are defined by positive and negative double precision values relative to the start coordinate point \((x_1, y_1)\). For instance, a negative value of \(H\) would place the end point at \(y_1 - H\).

\[
\begin{align*}
&\langle x_1 \ y_1 \ x_2 \ y_2 \ W \ \theta(x_1,y_1) \ \text{sBend}\rangle \\
&\langle x_1 \ y_1 \ L \ H \ W \ \theta(x_1,y_1) \ \text{sBendLH}\rangle
\end{align*}
\]

Figure 2.116: An example S-Bend GDS shape illustrating various parameters from the two \texttt{sBend} constructors.
2.9.3.5 S-Bend Inverse

Similar to S-Bends (section 2.9.3.4), \texttt{sBendInv} constructor is used to create s-bend waveguides using a positive tone resist. Alternatively, using a negative tone resist, a slot-waveguide of width $W + W_e$ and a slot of width $W$ is constructed. The shape is characterized by the start point $(x_1, y_1)$, length ($L$), height ($H$), waveguide width ($W$), exposure sleeve width ($W_e$) and the rotation about the start point $(x_1, y_1)$. \texttt{shapeReso} parameter defines the rendering resolution of the S-bend (Bezier based) curve. Both $L$ and $H$ parameters are defined by positive and negative double precision values relative to the start coordinate point $(x_1, y_1)$. For instance, a negative value of $H$ would place the end point $y$ value at $y_1 - H$.

$$<x_1 \ y_1 \ L \ H \ W \ W_e \ \theta_{(x_1,y_1)} \ sBendInv>$$

![Diagram](image)

\textbf{Figure 2.117: Schematic illustration showing various parameters from the \texttt{sBendInv} constructor.}
2.9.3.6 S-Bend Inverse Slot

Similar to S-Bends (section 2.9.3.4), \texttt{sBendInvSlot} constructor is used to create s-bend slot waveguides using a positive tone resist. The shape is characterized by the start point \((x_1, y_1)\), length \((L)\), height \((H)\), slot width \((W_s)\), gap \((g)\), exposure sleeve width \((W_e)\) and the rotation about the start point \((x_1, y_1)\) parameters. \texttt{shapeReso} parameter defines the rendering resolution of the S-bend (Bezier based) curve. Both \(L\) and \(H\) parameters are defined by positive and negative double precision values relative to the start coordinate point \((x_1, y_1)\). For instance, a negative value of \(H\) would place the end point \(y\) value at \(y_1 - H\).

\[
<x_1 \ y_1 \ L \ H \ W_s \ g \ W_e \ \theta_{(x_1, y_1)} \ \texttt{sBendInvSlot}>
\]

![Diagram showing various parameters from the sBendInvSlot constructor.](image)

\textbf{Figure 2.118:} Schematic illustration showing various parameters from the \texttt{sBendInvSlot} constructor.
2.9.3.7 Y-Bend

Y-Bends are constructed by merging two S-Bend curves. Similarly to the S-Bend curve defined in the section 2.9.3.4, Y-Bends are characterized by two constructors. First consists of the start $(x_1, y_1)$ and two end $(x_2, y_2)$ and $(x_3, y_3)$ points. The second is characterized by the start point $(x_1, y_1)$, pair of length $(L_2$ and $L_3)$ and pair of height $(H_2$ and $H_3)$ parameters. Both constructors are characterized by a waveguide width $(W)$ and the rotation about the start point $(x_1, y_1)$. shapeReso parameter defines the rendering resolution of the Y-bend (Bezier based) curve. The length and height parameters are defined by positive and negative double precision values relative to the start coordinate point $(x_1, y_1)$. For instance, a negative value of $H_3$ would place the end point $y$ value at $y_1 - H_3$. Figure 2.119 shows a negative $H_3$ value.

\[
\begin{align*}
&<x_1 \ y_1 \ x_2 \ y_2 \ x_3 \ y_3 \ W \ \theta_{(x_1,y_1)} \ yBend> \\
&<x_1 \ y_1 \ L_2 \ H_2 \ L_3 \ H_3 \ W \ \theta_{(x_1,y_1)} \ yBendLH>
\end{align*}
\]

![Diagram of Y-Bend](image)

Figure 2.119: An example Y-Bend GDS shape illustrating various parameters from the two yBend constructors.
2.9.3.8 Y-Bend Inverse

Similar to Y-Bends (section 2.9.3.7), yBendInv constructor is used to create a y-bend waveguide bifurcation using a positive tone resist. Alternatively, using a negative tone resist, a slot-waveguide of width \( W + W_e \) and a slot of width \( W \) is constructed. The shape characterized is characterized by the start coordinate point \((x_1, y_1)\), a pair of lengths \((L_2 \text{ and } L_3)\), a pair of heights \((H_2 \text{ and } H_3)\), waveguide width \( W \), exposure sleeve \( W_e \) and the rotation about the start point \((x_1, y_1)\) parameters. shapeReso parameter defines the rendering resolution of the Y-bend (Bezier based) curve. Both length and height parameters are defined by positive and negative double precision values relative to the start coordinate point \((x_1, y_1)\). For instance, a negative value of \( H_3 \) would place the end point \( y \) value at \( y_1 - H_3 \), hence value of \( H_3 \) value in Figure 2.120 is negative.

\[
<x_1 \ y_1 \ L_2 \ H_2 \ L_3 \ H_3 \ W \ W_e \ \theta_{(x_1,y_1)} \ yBendInv>
\]

Figure 2.120: Schematic illustration showing various parameters from the yBendInv constructor.
2.9.3.9 Y-Bend Inverse Slot

Similar to Y-Bends (section 2.9.3.7), yBendInv constructor is used to create a y-bend slot waveguide bifurcation using a positive tone resist. The shape characterized is determined by the start coordinate point \((x_1, y_1)\), a pair of lengths \((L_2, L_3)\), a pair of heights \((H_2, H_3)\), slot width \((W_s)\), gap \((g)\), exposure sleeve width \((W_e)\) and the rotation about the start point \((x_1, y_1)\) parameters. `shapeReso` parameter defines the rendering resolution of the Y-bend (Bezier based) curve. Both length and height parameters are defined by positive and negative double precision values relative to the start coordinate point \((x_1, y_1)\). For instance, a negative value of \(H_3\) would place the end point \(y\) value at \(y_1 - H_3\), hence value of \(H_3\) value in Figure 2.121 is negative.

\[
<x_1 \ y_1 \ L_2 \ H_2 \ L_3 \ H_3 \ W_s \ g \ W_e \ \theta_{(x_1, y_1)} \ yBendInvSlot>
\]

Figure 2.121: Schematic illustration showing various parameters from the yBendInvSlot constructor.
2.9.3.10 Y-Bend - 90 degree

\[ <x \ y \ r_1 \ r_2 \ w \ N_{sides} \ \theta_{(x,y)} \ yBend90> \]

Figure 2.122: An example of a 90 degree Y-Bend.
2.9.3.11 Y-Bend Inverse - 90 degree

\[ \langle x \ y \ r_1 \ r_2 \ w \ w_e \ N_{\text{sides}} \ \theta_{(x,y)} \ \text{yBendInv90} \rangle \]

Figure 2.123: An example of a 90 degree Y-Bend inverse shape.
2.9.3.12 Y-Bend Inverse Slot - 90 degree

\[<x \ y \ r_1 \ r_2 \ w_s \ g \ w_e \ N_{sides} \ \theta_{(x,y)} \ yBendInvSlot90>\]

Figure 2.124: An example of a 90 degree Y-Bend inverse slot shape.
2.9.3.13 90 Degree Bend

90 Degree Bends are Bezier curves characterized by two constructors. First consists of the start $(x_1, y_1)$ and end $(x_2, y_2)$ points. The second is characterized by the start point $(x_1, y_1)$, length ($L$) and height ($H$) parameters. Both constructors are characterized by a waveguide width ($W$) and the rotation about the start point $(x_1, y_1)$. `shapeReso` parameter defines the rendering resolution of the 90 Degree Bend (Bezier-based) curve.

```
<x1 y1 x2 y2 W \theta(x1,y1) 90degreeBend>
<x1 y1 L H W \theta(x1,y1) 90degreeBendLH>
```

![Diagram of a 90 degree bend GDS shape illustrating various parameters from the two constructors.](image)

Figure 2.125: An example 90 degree bend GDS shape illustrating various parameters from the two constructors.
2.9.3.14 90 Degree Bend Inverse

Similar to 90 Degree Bends (section 2.9.3.13), `90degreeBendInv` constructor is used to create a 90 degree bend waveguide structure using a positive tone resist. Alternatively, using a negative tone resist, a slot-waveguide of width $W + W_e$ and a slot of width $W$ is constructed. The shape is characterized by the start coordinate point $(x_1, y_1)$, length ($L$), height ($H$), waveguide width ($W$), exposure sleeve ($W_e$) and the rotation about the start point $(x_1, y_1)$ parameters. The `shapeReso` parameter defines the rendering resolution of the 90 degree (Bezier based) curve. Both length and height parameters are defined by positive and negative double precision values relative to the start coordinate point $(x_1, y_1)$. For instance, a negative value of $H$ would place the end point $y$ value at $y_1 - H$, hence value of $H$ in Figure 2.126 is positive.

$$
<x_1 \ y_1 \ L \ H \ W \ W_e \ \theta(x_1,y_1) \ 90degreeBendInv>
$$

Figure 2.126: Schematic illustration showing various parameters from the 90degreeBendInv constructor.
2.9.3.15 90 Degree Bend Inverse Slot

Similar to 90 Degree Bends (section 2.9.3.13), \texttt{90degreeBendInv} constructor is used to create a 90 degree bend slot waveguide structure using a positive tone resist. The shape characterized is characterized by the start coordinate point \((x_1, y_1)\), length \((L)\), height \((H)\), slot width \((W_s)\), gap \((g)\), exposure sleeve width \((W_e)\) and the rotation about the start point \((x_1, y_1)\) parameters. \texttt{shapeReso} parameter defines the rendering resolution of the 90 degree (Bezier-based) curve. Both length and height parameters are defined by positive and negative double precision values relative to the start coordinate point \((x_1, y_1)\). For instance, a negative value of \(H\) would place the end point \(y\) value at \(y_1 - H\), hence value of \(H\) in Figure 2.127 is positive.

\[
\langle x_1 \quad y_1 \quad L \quad H \quad W \quad g \quad W_e \quad \theta_{(x_1,y_1)} \quad \texttt{90degreeBendInvSlot}\rangle
\]

\[\text{Figure 2.127: Schematic illustration showing various parameters from the } \texttt{90degreeBendInvSlot} \text{ constructor.}\]
2.9.3.16 180 Degree Bend

The 180 degree, u-shaped, bend is characterized by lengths $L_1$ and $L_2$ with a coupled half torus of diameter $D$. The number of vertices constructing the torus is represented by the $N$ parameter. $D$ is measured from the midpoints of the two length segments.

$$<x \ y \ L_1 \ L_2 \ D \ W \ N \ \theta_{(x,y)} \ 180\text{degreeBend}>$$

![Diagram](Figure 2.128: 180 degree bend example.)
2.9.3.17 180 Degree Bend Inverse

Similar to the 180 degree bend in the previous section (2.9.3.16), the inverse device has an additional exposure sleeve parameter $W_c$.

\[
\begin{array}{cccccc}
<x & y & L_1 & L_2 & D & W & W_c & N & \theta(x,y) & 180\text{degreeBendInv}>
\end{array}
\]

Figure 2.129: 180 degree bend inverse example.
2.9.3.18 180 Degree Bend Inverse Slot

Similar to the 180 degree inverse bend in the previous section (2.9.3.17), the inverse slot device has an additional gap parameter $g$.

$$<x \ y \ L_1 \ L_2 \ D \ W \ g \ W_e \ N \ \theta_{(x,y)} \ 180\text{degreeBendInvSlot}>$$

![Figure 2.130: 180 degree bend inverse slot example.](image-url)
2.9.4 Racetrack

Racetrack object is characterized by the center coordinate \((x, y)\), length of the straight section \((L)\), track width \((W)\), inner radius \((r_{\text{in}})\), rotation about the center point \((\theta_{(x,y)})\), and the number of segments creating each of the curved sections \((N)\).

\[
<x \ y \ L \ W \ r_{\text{in}} \ \theta_{(x,y)} \ N \ \text{raceTrack}>
\]

Figure 2.131: An example GDS shape illustrating various parameters from the race-Track constructors. Dots along the structural periphery represent the number of points \((N)\) used to construct the curved racetrack sections.
2.9.5 Spiral Delay Line

Interdigitated spiral delay line objects are characterized by either uniform (Archimedes) and converging (Fermat’s) spacing between subsequent turns. Constant spacing Archimedes spiral delay line illustrated in Figure 2.9.5a is represented by

\[
r(\theta) = \begin{cases} 
  m\theta \\
  -m\theta 
\end{cases}
\]

(2.12)

where \( m = (S + W)/(2\pi) \), and the separation parameter \( S \) defines the pitch between subsequent spiral turns. Spiral is further defined by \( \Delta R \) defining the resolution of the spiral, the length of the coupling waveguide \( (L) \), the rotation about center \( (\theta_{(x,y)}) \), and \( EC = 1 \) or 0 corresponding to the waveguide coupling segment without and with semi-circular endcaps. Fermat’s spiral delay line illustrated in Figure 2.9.5b is represented by

\[
\begin{align*}
  r(\theta) &= \sqrt{a^2 + \theta} \\
  &= \sqrt{a^2 - \theta}
\end{align*}
\]

(2.14)

(2.15)

\( x \ y \ W \ N_{\text{turns}} \ S \ \Delta R \ L \ \theta_{(x,y)} \ EC \ \text{spiralDelayLineArch} \)

\( x \ y \ W \ N_{\text{turns}} \ a \ \Delta R \ L \ \theta_{(x,y)} \ EC \ \text{spiralDelayLineFermat} \)

Figure 2.9.5: Spiral delay lines for the (a) Archimedes spiral constructor \text{spiralDelayLineArch} \text{spiralDelayLineArch} and (b) Fermat spiral constructor \text{spiralDelayLineFermat} \text{spiralDelayLineFermat}. Central dot represents the coordinate defined by \( (x, y) \).

IMPORTANT NOTE: The above interdigitated spirals are created and fractured along the \( y \)-axis. On the one hand, this action eliminates any possibility of exceeding maximum number of allowed points per GDS shape. On the opposite
end of the spectrum, this action is computation intensive, consuming large amounts of RAM and takes a considerable amount of time for spirals of many turns defined by a large collection points.

The following is an interdigitated Archimedes spiral delay line that alleviates some of the computational strains of the previous version. The delay line illustrated in Figure 2.133 is characterized by a center point \((x, y)\), spiral width, the number of turns \((N_{\text{turns}})\), separation \((S)\) between subsequent turns, points per turn \((P_T)\), and the number of skipped turns before spiral rendering initiates \((S_T)\). In this case, points per spiral turn are controlled, hence for spirals containing many turns, the resolution of the outermost ring will be coarser than the internal rings. This spiral is generated much faster than ones shown in Figure 2.132.

\[
x y W N_{\text{turns}} S P_T S_T L \theta_{(x,y)} EC \text{ spiralDelayLineArchV2}
\]

Figure 2.133: Archimedes interdigitated spiral delay line constructed by \texttt{spiralDelayLineArchV2} constructor. Parameter \(S_T\) is an integer \((S_T \geq 0)\) defining the number of skipped turns before rendering of the spiral initiates.
2.9.6 Inverse Spiral Delay Line

Inverse spiral delay line objects are characterized by either uniform (Archimedes) and converging (Fermat’s) spacing between subsequent turns. Section 2.9.5 shows details for the construction of the two spirals. These spirals are defined by the center point \((x, y)\), slot width \(W\), exposure sleeve width \(W_e\), number of turns \(N_{\text{turns}}\), separation \(S\) between turns, and the \(\Delta R\) value defining resolution rendering of the spiral.

\[
\begin{align*}
  x & \quad y & \quad W & \quad W_e & \quad N_{\text{turns}} & \quad S & \quad \Delta R & \quad L & \quad \theta_{(x,y)} & \quad EC & \quad \text{spiralDelayLineArchInv} \\
  x & \quad y & \quad W & \quad W_e & \quad N_{\text{turns}} & \quad a & \quad \Delta R & \quad L & \quad \theta_{(x,y)} & \quad EC & \quad \text{spiralDelayLineFermatInv}
\end{align*}
\]

Figure 2.134: Inverse spiral delay lines for the (a) Archimedes spiral constructor \texttt{spiralDelayLineArchInv} and (b) Fermat spiral constructor \texttt{spiralDelayLineFermatInv}. Central dot represents the coordinate defined by \((x, y)\), \(W\) and \(W_e\) are the corresponding slot and exposure sleeve widths.
The following is another version of the inverted Archimedes spiral delay line. The delay line illustrated in Figure 2.135 is characterized by a center point \((x, y)\), spiral width, exposure sleeve width, the number of turns \(N_{\text{turns}}\), separation \(S\) between subsequent turns, points per turn \(P_T\), and the number of skipped turns before spiral rendering initiates \(S_T\). In this case, points per spiral turn are controlled, hence for spirals containing many turns, the resolution of the outermost ring will be coarser than the internal rings.

\[
\begin{align*}
  x & \quad y & \quad W & \quad W_e & \quad N_{\text{turns}} & \quad S & \quad P_T & \quad S_T & \quad L & \quad \theta(x,y) & \quad EC & \quad \text{spiralDelayLineArchV2Inv}
\end{align*}
\]

Figure 2.135: Archimedes interdigitated spiral delay line constructed by \texttt{spiralDelayLineArchV2Inv} constructor. Parameter \(S_T\) is an integer \((S_T \geq 0)\) defining the number of skipped turns before rendering of the spiral initiates.

**IMPORTANT NOTE:** The above interdigitated spirals (Figures 2.134-2.135) are created and fractured along the \(y\)-axis. As mentioned in the previous section, fracturing eliminates any possibility of exceeding maximum number of allowed points per GDS shape. Unfortunately, the fracturing is computationally intensive, consuming large amounts of RAM for spirals of many turns defined by a large collection points.
2.9.7 Gratings

2.9.7.1 Grating

Grating structure is defined by the line width, length and pitch. A single line is cast into a GDS structure `singleRectangleStructName` and arrayed (instantiated) at the specified pitch within the existing GDS structure. \( N_{\text{lines}} \) parameter represents the number of instantiated lines.

\[
\begin{align*}
\langle \text{singleRectangleStructName} &\ x\ y\ \text{W} \ \text{L} \ \text{Pitch} \ \text{N}_{\text{lines}} \ \text{grating}\rangle \\
\end{align*}
\]

Figure 2.136: Example of the \texttt{grating} constructor.
2.9.7.2 Apodized Grating

The apodized grating structure is defined by the grating length ($G_L$), the increment ($\Delta G_L$), grating line height ($H$), duty cycle cutoff ($d_{CC}$), and grating parameters $p_i$ and $d_j$. The grating extends from ($-\frac{G_L}{2}$) to ($+\frac{G_L}{2}$) with the lower edge centered around $(x,y)$.

$$<x\ y\ G_L\ \Delta G_L\ \ H\ \ d_{CC}\ \ p_1\ \ p_2\ \ p_3\ \ p_4\ \ d_1\ \ d_2\ \ d_3\ \ d_4\ \ \theta_{(x,y)}\ \text{apodizedGrating}>$$

![Apodized Grating Diagram](image)

Figure 2.137: Example of the `apodizedGrating` constructor.
2.9.7.3 Grating Coupler

Structure characterized by a center \((x, y)\), distance \((r)\) from center to the mid-point of the first arc section, width \((W)\) and pitch \((p)\) of the arc section, start \((\theta_s)\) and end angles \((\theta_e)\), number of sides \((N_{\text{sides}})\) for each of the arc sections and number of elements \((N_{\text{elements}})\).

\[
\langle x \ y \ r \ W \ p \ \theta_s \ \theta_e \ N_{\text{sides}} \ N_{\text{elements}} \ \text{gratingCoupler}\rangle
\]

Figure 2.138: Example illustrating various parameters of the gratingCoupler constructor.
2.9.7.4 Grating Couplers With Waveguides

The following constructors generate grating couplers with integrated waveguides within the $x - y$ plane using

$$q\lambda_o = n_{eff}\sqrt{x^2 + y^2 - xn_c\cos\theta_c}$$

(2.16)

where the focal point is located at the origin $(x, y)$ with $x$ along the waveguide axis, $q$ is an integer for each grating line, $\theta_c$ is the grating angle span, $n_c$ and $n_{eff}$ are the respective cladding and effective refractive index values, and $\lambda_o$ is the wavelength. The following are the constructors for the waveguide (GDS Layer $L_{wg}$) with an integrated grating coupler (GDS Layer $L_g$)

```
<x y w L H s λ_o n_{eff} n_c θ_c R g_p ratio N_g N_s L_{wg} L_g EC θ(x,y) gratingCWGinv>
```

```
<x y w L H λ_o n_{eff} n_c θ_c R g_p ratio N_g N_s L_{wg} L_g EC θ(x,y) gratingCWG>
```

As labeled in Figures 2.139 and 2.140, the structure is placed and rotated $(\theta(x,y))$ relative to the origin $(x, y)$. Waveguides are constructed using a bezier s-bend curve and defined by a width ($w$), length ($L$) and height ($H$). A negative value of $H$ will extend the waveguide below the $y$-axis. Sleeve width for inverse waveguides (Figures 2.139a and 2.140a) is defined by $s$, $R$ is the distance to the outermost grating, $g_p$ is the grating pitch and the duty cycle is defined by the ratio value. Consequently the grating line width is given by

$$g_w = g_p * ratio$$

(2.17)

$N_g$ defines the number of generated grating lines with the maximum number grating lines within $R$ given by

$$N_{max} = ceil \left(\frac{n_{eff} R}{\lambda_o}\right)$$

(2.18)

The outer and inner radius of each ring is defined by

$$r_{q(out)} = \frac{q\lambda_o}{n_{eff} - n_c \cos \alpha_i \cos \theta}$$

(2.19)

$$r_{q(in)} = r_{q(out)} - (g_p * ratio) = r_{q(out)} - g_w$$

(2.20)

where

$$\theta = \arccos \left(\frac{n_{eff} - \lambda_o}{g_p}\right)$$

(2.21)

and $\alpha_i$ is the angular range from $-\theta_c$ to $\theta_c$. Parameter $(x, y, w, L, H, s, \lambda_o, R, g_p)$ dimensions are expressed in micrometers.
Figure 2.139: Grating couplers (a) waveguide with sleeve and (b) waveguide with $EC = 0$. 
Figure 2.140: Grating couplers with endcaps. (a) waveguide with sleeve and (b) waveguide with EC = 1.
2.9.8 Photonic Crystals and Hexagonal Arrays

Photonic crystal and hexagonal array structures characterized by a center \((x,y)\), circle radius \(r\), pitch \((x_p)\), number of elements \(N_x\) and \(N_y\) in the corresponding \(x\) and \(y\) directions, and spacing \((\delta)\) between the two arrays. Parameter \texttt{uniqueStructName} represents the name for the structure (cell) for the circular element. The circular element is then arrayed to construct the photonic crystal structure.

Within constructors \texttt{phC} and \texttt{hex}, parameter \(N_{sides}\) defines the number of sides of the individual polygon/circular apertures. When using vectorized constructors (\texttt{phCV} and \texttt{hexV}), resulting structures are circles with number of sides determined by the shape rendering resolution parameter \texttt{shapeReso}.

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ N_{sides} \ \delta \ \texttt{phC}>
\]

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ \delta \ \texttt{phCV}>
\]

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ N_{sides} \ \texttt{hex}>
\]

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ \texttt{hexV}>
\]

\(N_x\) and \(N_y\) in the corresponding \(x\) and \(y\) directions, and spacing \((\delta)\) between the two arrays. Parameter \texttt{uniqueStructName} represents the name for the structure (cell) for the circular element. The circular element is then arrayed to construct the photonic crystal structure.

Within constructors \texttt{phC} and \texttt{hex}, parameter \(N_{sides}\) defines the number of sides of the individual polygon/circular apertures. When using vectorized constructors (\texttt{phCV} and \texttt{hexV}), resulting structures are circles with number of sides determined by the shape rendering resolution parameter \texttt{shapeReso}.

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ N_{sides} \ \delta \ \texttt{phC}>
\]

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ \delta \ \texttt{phCV}>
\]

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ N_{sides} \ \texttt{hex}>
\]

\[
<\texttt{uniqueStructName} \ x \ y \ r \ x_p \ N_x \ N_y \ \texttt{hexV}>
\]

\textbf{Figure 2.141:} (a) Photonic crystal and (b) hexagonal array examples illustrating various constructor parameters.
2.9.9 Disc-Ring Architectures

2.9.9.1 Disc-Ring - Bezier Curves, Arcs and Endcaps

The following sections illustrate coupled waveguides to disc and ring structures. The coupling region is defined by an arc section that symmetrically interacts with the disc/ring structure. The waveguide region leading away from the coupling region is defined by either a Bezier or an arc segment.

Figure 2.142: Schematic illustration highlighting the waveguide coupling region constructed using (a) Bezier and (b) arc segments.

Figure 2.142a illustration has coupling region (labeled as 1) defined by an arc. The continuing segment ending at a distance $L$ is defined by a Bezier curve consisting of start and end points ($P_1$ and $P_2$) and their corresponding control points $C_1$ and $C_2$. Control points are defined as,

\begin{align*}
C_{1x} &= P_{1x} + \frac{R}{4} \cdot \cos(\phi) \\
C_{1y} &= P_{1y} - \frac{R}{4} \cdot \sin(\phi) \\
C_{2x} &= P_{2x} - \frac{L}{2} \\
C_{2y} &= P_{2y}
\end{align*}  

(2.22a-d)

where $R = \sqrt{H^2 + L^2}$, and $\phi = 90 - \theta$.

Figure 2.142b shows a coupling region defined by two arc segments. For this segment, the straight waveguide portion of length $L$ is omitted. In both cases, the drawn structures are mirrored around the $y$-axis.
Figure 2.143: Illustrations showing semicircular encasps on the right hand side of (a) a waveguide and (b) an inverse and inverse positive waveguide. Endcaps are included on both left and right sides of the waveguide when parameter $EC = 1$. The red circle denotes the end point of the waveguide structure. Semicircular endcap radius of the equals half of the waveguide width.

Figure 2.143 shows shows waveguide structures with semicircular endcaps. The structures are encountered in subsequent sections and are useful when stressed thin film underlayers are patterned using thick resists. The corner rounding at the waveguide termination end alleviates the stress field, thereby mitigating resist and thin film cracking during subsequent reactive ion etch processing steps.
2.9.9.2 Disc-Ring Infinite

Shapes characterized by a straight waveguide \((R_{wg} = \infty)\) above a disc and ring objects. Both objects are described by the center coordinate \((x, y)\), number of segments \((N)\), gap between the object and shape \((g)\), waveguide length \((L)\) and width \((W)\). Disc object is defined by a radius \((r_d)\) and the ring is characterized by an inner radius \(r_i\) and ring width \((W_r)\). Boolean parameter \(EC\) controls if semicircular endcaps are incorporated in the waveguide structure. \(EC\) is an integer value with zero representing false, hence waveguides without endcaps, i.e. \(EC = 0\) and \(EC = 1\) for waveguides without and with endcaps, respectively.

\[
\begin{align*}
\langle x & \ y \ r_d \ N \ g \ L \ W \ EC \ disclnfinite > \\
\langle x & \ y \ r_i \ W_r \ N \ g \ L \ W \ EC \ ringlnfinite >
\end{align*}
\]

![Diagram of Disc-Ring Infinite Shapes](image)

**Figure 2.144**: Examples illustrating various parameters of the (a) \(\text{disclnfinite}\) and (b) \(\text{ringlnfinite}\) constructors.
Infinite disc and ring structures with an additional coupling waveguide.

\[
\begin{align*}
&\langle x \ y \ r_d \ N \ g_1 \ L_1 \ W_1 \ g_2 \ L_2 \ W_2 \ EC \ \text{discInfDS}\rangle \\
&\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_1 \ W_1 \ g_2 \ L_2 \ W_2 \ EC \ \text{ringInfDS}\rangle
\end{align*}
\]

Figure 2.145: Examples illustrating various parameters of the (a) discInfDS and (b) ringInfDS constructors.
2.9.9.3 Disc-Ring Infinite Inverse

The structure is similar to the disc-ring waveguide infinite structure defined in the previous section 2.9.9.2. Here, a slot waveguide of width \( W \) is formed by exposing a surrounding rectangular region of width \( W_e \). The overall shapes are characterized by a straight waveguide \((R_{wg} = \infty)\) above a disc and ring objects. Both objects are described by the center coordinate \((x, y)\), number of segments \((N)\), gap between the object and shape \((g)\), waveguide length \((L)\), waveguide width \((W)\) and exposure sleeve width \((W_e)\). Disc object is defined by a radius \((r_d)\) and the ring is characterized by an inner radius \((r_i)\) and ring width \((W_r)\). Boolean parameter \(EC\) controls if semicircular endcaps are incorporated in the waveguide structure \((EC = 0\) and \(EC = 1\) for waveguides without and with endcaps, respectively).

\[
\begin{align*}
<x & \ y & \ r_d & \ N \ g \ L \ W \ W_e \ EC \ disclnfiniteInv> \\
<x & \ y & \ r_i & \ W_r \ N \ g \ L \ W \ W_e \ EC \ ringlnfiniteInv>
\end{align*}
\]

Figure 2.146: Examples illustrating various parameters of the (a) disclnfiniteInv and (b) ringlnfiniteInv constructors.
Infinite inverse disc and ring structures with an additional waveguide.

\[
<x \ y \ r_d \ N \ g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ discInvDS>
\]
\[
<x \ y \ r_i \ W_r \ N \ g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ ringInvDS>
\]

Figure 2.147: Examples illustrating various parameters of the (a) discInvDS and (b) ringInvDS constructors.
2.9.9.4 Disc-Ring Infinite Inverse Positive Tone

The structure is similar to the disc-ring waveguide structure defined in section 2.9.9.2. Here, a waveguide of width $W$ and either a ring of width $W_r$ or disc of radius $r_d - r_e$ are formed by positive tone resist exposure of the respective exposure sleeve regions $W_e$ and $r_e$. The overall shapes are characterized by a straight waveguide ($R_{wg} = \infty$) above a disc and ring objects. Both objects are described by the center coordinate $(x, y)$, number of segments ($N$), gap between the object and shape ($g$), waveguide length ($L$), waveguide width ($W$) and exposure sleeve width ($W_e$). Disc object is defined by a radius ($r_d$) and the ring is characterized by a radius ($r_i$) and ring width ($W_r$).

$$\langle x \ y \ r_d \ r_e \ N \ g \ L \ W \ W_e \ EC \ discInfiniteInvPos \rangle$$

$$\langle x \ y \ r_i \ W_r \ r_e \ N \ g \ L \ W \ W_e \ EC \ ringInfiniteInvPos \rangle$$

Figure 2.148: Examples illustrating various parameters of the (a) discInfiniteInvPos and (b) ringInfiniteInvPos constructors.
Infinite inverse positive disc and ring structures with an additional coupling waveguide.

\[ \langle x \ y \ r_d \ r_e \ N \ g_1 \ L_1 \ W_1 \ W_{c1} \ g_2 \ L_2 \ W_2 \ W_{c2} \ EC \ disclnInvPosDS \rangle \]

\[ \langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_1 \ W_1 \ W_{c1} \ g_2 \ L_2 \ W_2 \ W_{c2} \ EC \ ringlnInvPosDS \rangle \]

Figure 2.149: Examples illustrating various parameters of the (a) disclnInvPosDS and (b) ringlnInvPosDS constructors.
2.9.9.5 Disc-Ring Symmetric Bezier

Shapes characterized by a curved waveguide above a disc \(R_{\text{waveGuide}} = -R_{\text{Disc}}\) and ring \(R_{\text{waveGuide}} = -R_{\text{Ring}}\) objects. Both objects (disc/ring) are described by the center coordinate \((x, y)\), number of segments \((N)\), gap between the object and waveguide \((g)\), waveguide opening angle \((\theta)\), number of sides \((N_{\text{wg}})\), width \((W)\), length \((L)\), height \((H)\), disc radius \((r_d)\), ring radius \((r_i)\) and width \((W_r)\). \(EC = 0\) and \(EC = 1\) for waveguides without and with endcaps, respectively.

\[
\begin{align*}
\langle x &\ y &\ r_d &\ N &\ g &\ \theta &\ N_{\text{wg}} &\ W &\ L &\ H &\ EC \quad \text{discSymmetric} \rangle \\
\langle x &\ y &\ r_i &\ W_r &\ N &\ g &\ \theta &\ N_{\text{wg}} &\ W &\ L &\ H &\ EC \quad \text{ringSymmetric} \rangle
\end{align*}
\]

Figure 2.150: Example shapes illustrating various parameters from the (a) discSymmetric and (b) ringSymmetric constructors.
Disc symmetric structure with an additional coupling waveguide.

\[ \langle x \ y \ r_d \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ L_1 \ H \ g_2 \ W_2 \ L_2 \ EC \ discSymDS \rangle \]

Figure 2.151: Example shape illustrating various parameters from the discSymDS constructor.
Disc symmetric structure with an additional coupling pulley.

\[
<x \ y \ r_d \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ L_1 \ H_1 \ g_2 \ \theta_2 \ W_2 \ L_2 \ H_2 \ EC \ discSymPul> 
\]

**Figure 2.152:** Example shape illustrating various parameters from the *discSymPul* constructors.
Ring symmetric structure with an additional coupling waveguide.

Figure 2.153: Example shape illustrating various parameters from the \texttt{ringSymDS} constructor.
Ring symmetric structure with an additional coupling pulley.

\(<x \ y \ r_i \ W_r \ N \ g_1 \ \theta_1 \ N_wg \ W_1 \ L_1 \ H \ g_2 \ \theta_2 \ W_2 \ L_2 \ H_2 \ EC \ ringSymPul>\)

Figure 2.154: Example shape illustrating various parameters from the ringSymPul constructor.
The following disc-ring waveguide symmetric structures are defined by the coupling length parameter $L_c$.

\[
\begin{align*}
\langle x & \ y & \ r_d & \ N & \ g & \ L_c & \ N_{wg} & \ W & \ L & \ H & \ EC \ discSymmetricLC \rangle \\
\langle x & \ y & \ r_i & \ W_r & \ N & \ g & \ L_c & \ N_{wg} & \ W & \ L & \ H & \ EC \ ringSymmetricLC \rangle
\end{align*}
\]

\textit{Figure 2.155: Example shapes illustrating various parameters from the (a) discSymmetricLC and (b) ringSymmetricLC constructors.}
Disc symmetric structure with an additional coupling waveguide.

\[ \langle x \ y \ r_d \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ L_1 \ H \ g_2 \ W_2 \ L_2 \ EC \ discSymLCDS \rangle \]

Figure 2.156: Example shape illustrating various parameters from the discSymLCDS constructor.
Disc symmetric structure with an additional coupling pulley.

\[
\begin{align*}
&\langle x \ y \ r_d \ N \ g_1 \ L_{c1} \ N_wg \ W_1 \ L_1 \ H_1 \ g_2 \ L_{c2} \ W_2 \ L_2 \ H_2 \ EC \ discSymLCPul\rangle
\end{align*}
\]

**Figure 2.157:** Example shape illustrating various parameters from the `discSymLCPul` constructor.
Ring symmetric structure with an additional coupling waveguide.

$$<x \ y \ r_i \ W_r \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ L_1 \ H \ g_2 \ W_2 \ L_2 \ EC \ ringSymLCDS>$$

Figure 2.158: Example shape illustrating various parameters from the ringSymLCDS constructor.
Ring symmetric structure with an additional coupling pulley.

\[ <x \ y \ r_i \ W_r \ N_1 \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_1 \ H_1 \ g_2 \ L_{c2} \ W_2 \ L_2 \ H_2 \ EC \ ringSymLCPul> \]

Figure 2.159: Example shape illustrating various parameters from the \texttt{ringSymLCPul} constructor.
2.9.9.6 Disc-Ring Symmetric Arc

Symmetric structures similar to ones described in section 2.9.9.5 with coupling regions constructed using arcs. Both objects (disc/ring) are described by the center coordinate \((x, y)\), number of segments \((N)\), gap between the object and waveguide \((g)\), waveguide opening angle \((\theta)\), number of sides \((N_{wg})\), width \((W)\), and length of the connecting straight waveguide section \((L_s)\), disc radius, \((r_d)\), ring radius \((r_i)\) and width \((W_r)\). \(EC = 0\) and \(EC = 1\) for waveguides without and with endcaps, respectively.

\[
\begin{align*}
&<x \ y \ r_d \ N \ g \ \theta \ N_{wg} \ W \ L_s \ EC \ discSymmetricA> \\
&<x \ y \ r_i \ W_r \ N \ g \ \theta \ N_{wg} \ W \ L_s \ EC \ ringSymmetricA>
\end{align*}
\]

![Figure 2.160: Example shapes illustrating various parameters from the (a) discSymmetricA and (b) ringSymmetricA constructors.](image)
Disc symmetric arc structure with an additional coupling waveguide.

\(<x \ y \ rd \ N_1 \ g_1 \ N_{wg} \ W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ discSymADS>\)

**Figure 2.161**: Example shape illustrating various parameters from the *discSymADS* constructor.
Disc symmetric arc structure with an additional coupling pulley.

\[ <x \ y \ r_d \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ \theta_2 \ W_2 \ L_{S2} \ EC \ \text{discSymAPul}> \]

Figure 2.162: Example shape illustrating various parameters from the discSymAPul constructor.
Ring symmetric arc structure with an additional coupling waveguide.

\[
\begin{align*}
&<x \ y \ r_i \ W_r \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ \text{ringSymADS}>
\end{align*}
\]

**Figure 2.163:** Example shape illustrating various parameters from the `ringSymADS` constructor.
Ring symmetric arc structure with an additional coupling pulley.

\[
<x \ y \ r_i \ W_r \ N \ g_1 \ N_wg \ W \ L_S \ g_2 \ \theta_2 \ W_2 \ L_{S2} \ EC \ ringSymAPul>
\]

Figure 2.164: Example shape illustrating various parameters from the \textit{ringSymAPul} constructor.
The following arc based, disc-ring waveguide symmetric structures are defined by the coupling length parameter $L_c$.

$$<x \ y \ r_d \ N \ g \ L_c \ N_{wg} \ W \ L_S \ EC \ discSymmetricLCA>$$

$$<x \ y \ r_i \ W_r \ N \ g \ L_c \ N_{wg} \ W \ L_S \ EC \ ringSymmetricLCA>$$

![Diagram](image)

Figure 2.165: Example shapes illustrating various parameters from the (a) discSymmetricLCA and (b) ringSymmetricLCA constructors.
Disc symmetric arc structure with an additional coupling waveguide.

\[
\langle x \ y \ r_d \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ discSymLCADS \rangle
\]

Figure 2.166: Example shape illustrating various parameters from the (a) discSymLCADS constructor.
Disc symmetric arc structure with an additional coupling pulley.

\[
< x \ y \ r_d \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ L_{S2} \ EC \ discSymLCApul >
\]

Figure 2.167: Example shape illustrating various parameters from the (a) discSymLCApul constructor.
Ring symmetric arc structure with an additional coupling waveguide.

\[
\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ ringSymLCADS \rangle
\]

Figure 2.168: Example shape illustrating various parameters from the \textit{ringSymLCADS} constructor.
Ring symmetric arc structure with an additional coupling pulley.

\[
\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ L_{S2} \ EC \ ring\text{SymLCAPul}\rangle
\]

Figure 2.169: Example shape illustrating various parameters from the ring\text{SymLCAPul} constructor.
2.9.9.7 Disc-Ring Symmetric Inverse Bezier

The structure is similar to the disc-ring waveguide symmetric structure defined in the previous section (2.9.9.5). Here, a slot waveguide of width $W$ is formed by exposing a surrounding rectangular region of width $W_e$. Using a negative tone resist, the resulting structure would produce a disc or ring shape at a distance $g$ to a slotted waveguide.

\[
<x y r_d N g \theta N_{wg} W W_e L H EC \text{ discSymmetricInv}>
\]
\[
<x y r_i W_i N g \theta N_{wg} W W_e L H EC \text{ ringSymmetricInv}>
\]

Figure 2.170: Example shapes illustrating various parameters from the (a) discSymmetricInv and (b) ringSymmetricInv constructors.
Disc symmetric inverse structure with an additional coupling waveguide.

\[ <x \ y \ rd \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ W_{e1} \ L_1 \ H \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ discSymInvDS> \]

Figure 2.171: Example shape illustrating various parameters from the (a) discSymInvDS constructor.
Disc symmetric inverse structure with an additional coupling pulley.

\(<\!x\! y\! r_d\! N_g\! \theta_1\! N_{wg}\! W_1\! W_{e1}\! L_1\! H_1\! \theta_2\! W_2\! W_{e2}\! L_2\! H_2\! EC\! \text{discSymInvPul}\!>\)

![Diagram](image)

**Figure 2.172:** Example shape illustrating various parameters from the (a) \text{discSymInvPul} constructor.
Ring symmetric inverse structure with an additional coupling waveguide.

\[ \langle x, y, r_i, W_r, N, g_1, \theta, N_{wg}, W_1, W_{c1}, L_1, H, g_2, W_2, W_{c2}, L_2, EC \rangle \text{ ringSymInvDS} \]

**Figure 2.173:** Example shape illustrating various parameters from the `ringSymInvDS` constructor.
Ring symmetric inverse structure with an additional coupling pulley.

\[
\begin{align*}
<x \ y \ r_i \ W_r \ N \ g_1 \ N \ N_g \ W_1 \ L_1 \ H_1 \ g_2 \ W_2 \ L_2 \ H_2 \ EC \ ringSymInvPul>
\end{align*}
\]

Figure 2.174: Example shape illustrating various parameters from the \textit{ringSymInvPul} constructor.
The following disc-ring waveguide symmetric inverse structures are defined by the coupling length parameter $L_c$.

$$<x \ y \ r_d \ N \ g \ L_c \ N_wg \ W \ W_e \ L \ H \ EC \ discSymmetricInvLC>$$

$$<x \ y \ r_i \ W_r \ N \ g \ L_c \ N_wg \ W \ W_e \ L \ H \ EC \ ringSymmetricInvLC>$$

**Figure 2.175:** Example shapes illustrating various parameters from the (a) discSymmetricInvLC and (b) ringSymmetricInvLC constructors.
Disc symmetric inverse structure with an additional coupling waveguide.

\[ <x \ y \ rd \ N_1 \ N_{wg} \ W_1 \ W_{c1} \ L_1 \ H \ g_2 \ W_2 \ W_{c2} \ L_2 \ EC \ discSymInvLCDS> \]

Figure 2.176: Example shape illustrating various parameters from the (a) discSymInvLCDS constructor.
Disc symmetric inverse structure with an additional coupling pulley.

\[ <x \ y \ rd \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{c1} \ L_1 \ H_1 \ g_2 \ L_{c2} \ W_2 \ W_{c2} \ L_2 \ H_2 \ EC \ discSymInvLCPul> \]

Figure 2.177: Example shape illustrating various parameters from the (a) \texttt{discSymInvLCPul} constructor.
Ring symmetric inverse structure with an additional coupling waveguide.

\[
<x \ y \ r_i \ W_r \ N_1 \ L_c \ N_{wg} \ W_1 \ W_{c1} \ L_1 \ H \ g_2 \ W_2 \ W_{c2} \ L_2 \ EC \ ringSymInvLCDS>
\]

**Figure 2.178:** Example shape illustrating various parameters from the `ringSymInvLCDS` constructor.
Ring symmetric inverse structure with an additional coupling pulley.

\[<x \ y \ r_i \ W_r \ N \ g_1 \ L_{c1} \ N_w \ g \_1 \ W_1 \ L_{c1} \ H_1 \ g_2 \ L_{c2} \ W_2 \ L_{c2} \ H_2 \ EC \ ring\text{SymInvLCPul}>\]

**Figure 2.179:** Example shape illustrating various parameters from the `ringSymInvLCPul` constructor.
2.9.9.8 Disc-Ring Symmetric Inverse Arc

Symmetric inverse structures constructed using arcs are similar to ones described in section 2.9.9.6. Here, a slot waveguide of width $W$ is formed by exposing a surrounding rectangular region of width $W_e$. Using a negative tone resist, the resulting structure would produce a disc or ring shape at a distance $g$ to a slotted waveguide.

$$<x \ y \ r_d \ N \ g \ \theta \ N_{wg} \ W \ W_e \ L_S \ EC \ discSymmetricInvA>$$

$$<x \ y \ r_i \ W \ N \ g \ \theta \ N_{wg} \ W \ W_e \ L_S \ EC \ ringSymmetricInvA>$$

![Diagrams of disc and ring structures](attachment:diagram.png)

Figure 2.180: Example shapes illustrating various parameters from the (a) discSymmetricInvA and (b) ringSymmetricInvA constructors.
Disc symmetric inverse arc structure with an additional coupling waveguide.

\[
\text{discSymInvADS}(x, y, r_d, N, g_1, \theta, N_{wg}, W_1, W_{e1}, L_S, g_2, W_2, W_{e2}, L_2, EC)
\]

**Figure 2.181:** Example shape illustrating various parameters from the `discSymInvADS` constructor.
Disc symmetric inverse arc structure with an additional coupling pulley.

\[
\langle x\ y\ r_d\ N\ g_1\ \theta_1\ N_{wg}\ W_1\ L_{S_1}\ g_2\ \theta_2\ W_2\ L_{S_2}\ EC\ \text{discSymInvAPul}\rangle
\]

**Figure 2.182:** Example shape illustrating various parameters from the `discSymInvAPul` constructor.
Ring symmetric inverse arc structure with an additional coupling waveguide.

\[
\begin{align*}
<x \ y \ r_i \ W_r \ N \ g_1 \ \theta \ N_{wg} \ W_e \ L_S \ g_2 \ W_2 \ W_{c2} \ L_2 \ EC \ ringSymInvADS>
\end{align*}
\]

Figure 2.183: Example shapes illustrating various parameters from the ringSymInvADS constructor.
Ring symmetric inverse arc structure with an additional coupling pulley.

\[ x \ y \ r_i \ W_r \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ \theta_2 \ W_2 \ L_{S2} \ EC \ \text{ringSymInvAPul} \]

Figure 2.184: Example shapes illustrating various parameters from the \text{ringSymInvAPul} constructor.
The following arc based, disc-ring waveguide symmetric inverse structures are defined by the coupling length parameter $L_c$.

\[ <x, y, r_d, N, g, L_c, N_{wg}, W, W_e, L_S, EC \text{ discSymmetricInvLCA}> \]

\[ <x, y, r_i, W_r, N, g, L_c, N_{wg}, W, W_e, L_S, EC \text{ ringSymmetricInvLCA}> \]

\[ W \]
\[ W_e \]
\[ L_c \]
\[ L_S \]

Figure 2.185: Example shapes illustrating various parameters from the (a) discSymmetricInvLCA and (b) ringSymmetricInvLCA constructors.
Disc symmetric inverse arc structure with an additional coupling waveguide.

\[ <x \ y \ r_d \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{c1} \ L_S \ g_2 \ W_2 \ W_{c2} \ L_2 \ EC \ discSymInvLCADS> \]

Figure 2.186: Example shape illustrating various parameters from the `discSymInvLCADS` constructor.
Disc symmetric inverse arc structure with an additional coupling pulley.

\[
<x \ y \ r_d \ N \ g_1 \ L_c \ Nbg \ W_1 \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ L_{S2} \ EC \ discSymInvLCApul>
\]

Figure 2.187: Example shape illustrating various parameters from the discSymInvLCApul constructor.
Ring symmetric inverse arc structure with an additional coupling waveguide.

\[
\begin{align*}
\langle x, y, r, W, N, g_1, L_c, N_{wg}, W_1, W_{e1}, L_S, g_2, W_2, W_{e2}, L_2, EC \rangle & \text{ ringSymInvLCADS} \rangle
\end{align*}
\]

Figure 2.188: Example shape illustrating various parameters from the ringSymInvLCADS constructor.
Ring symmetric inverse arc structure with an additional coupling pulley.

\[<x\ y\ r_i\ W_r\ N\ g_1\ L_{c1}\ N_{w1}\ W_1\ L_{c2}\ W_2\ L_{c2}\ EC\ ringSymInvLCAPul>\]

Figure 2.189: Example shape illustrating various parameters from the ringSymInvLCA-Pul constructor.
2.9.9.9 Disc-Ring Symmetric Inverse Positive Tone Bezier

The structure is similar to the disc-ring waveguide symmetric structure defined in section 2.9.9.5. Here, a slot waveguide is formed at a distance $g$ away from a disc or a ring structure using a positive resist exposure of the exposure sleeve elements $W_e$ and $r_e$.

$<x\ y\ r_d\ \ r_e\ N\ g\ \ \theta\ N_{wg}\ W\ W_e\ L\ H\ EC\ discSymmetricInvPos>$$<x\ y\ r_i\ W_r\ r_e\ N\ g\ \ \theta\ N_{wg}\ W\ W_e\ L\ H\ EC\ ringSymmetricInvPos>$$

Figure 2.190: Example shapes illustrating various parameters from the (a) discSymmetricInvPos and (b) ringSymmetricInvPos constructors.
Disc symmetric inverse positive structure with an additional coupling waveguide.

<\(x\ y\ \rd\ \re\ N\ g_1\ \theta\ \N\ W_1\ \L_1\ H\ g_2\ W_2\ \L_2\ \EC\ discSymInvPosDS> 

Figure 2.191: Example shape illustrating various parameters from the discSymInvPosDS constructor.
Disc symmetric inverse positive structure with an additional coupling pulley.

\[<x, y, r_d, r_c, N, g_1, N_w, W_1, W_2, L_1, L_2, H_1, H_2, \theta_1, \theta_2, EC, \text{discSymInvPosPul}>\]

Figure 2.192: Example shape illustrating various parameters from the \text{discSymInvPosPul} constructor.
Ring symmetric inverse positive structure with an additional coupling waveguide.

<\begin{align*} x & y \\ r_1 & W_r \\ r_c & N \\ g_1 & \theta \\ Nwg & W_1 \\ W_{e1} & L_1 \\ H & g_2 \\ W_2 & W_{e2} \\ L_2 & EC \end{align*}> \text{ ringSymInvPosDS }>

Figure 2.193: Example shape illustrating various parameters from the ringSymInvPosDS constructor.
Ring symmetric inverse positive structure with an additional coupling pulley.

\(<x \ y \ r \ W \ r_e \ N \ N_{wg} \ W_1 \ W_{c1} \ W_2 \ L_1 \ L_2 \ H_1 \ H_2 \ g_1 \ g_2 \ \theta_1 \ \theta_2 \ ringSymInvPosPul>\)

Figure 2.194: Example shape illustrating various parameters from the ringSymInvPosPul constructor.
The following disc-ring waveguide symmetric inverse positive tone structures are defined by the coupling length parameter $L_c$.

$$\langle x \ y \ r_d \ r_e \ N \ g \ L_c \ N_wg \ W \ W_e \ L \ H \ EC \ discSymmetricInvPosLC \rangle$$

$$\langle x \ y \ r_i \ W_r \ r_e \ N \ g \ L_c \ N_wg \ W \ W_e \ L \ H \ EC \ ringSymmetricInvPosLC \rangle$$

Figure 2.195: Example shapes illustrating various parameters from the (a) discSymmetricInvPosLC and (b) ringSymmetricInvPosLC constructors.
Disc symmetric inverse positive structure with an additional coupling waveguide.

\[ <x \ y \ rd \ r_e \ N_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_1 \ H \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ discSymInvPosLCDS> \]

Figure 2.196: Example shape illustrating various parameters from the discSymInvPosLCDS constructor.
Disc symmetric inverse positive structure with an additional coupling pulley.

\[
<x y r_d r_c N g_1 N_w g_2 W_1 L_1 H_1 g_2 W_2 L_2 H_2 EC \text{discSymInvPosLCPul}>
\]

**Figure 2.197:** Example shape illustrating various parameters from the `discSymInvPosLCPul` constructor.
Ring symmetric inverse positive structure with an additional coupling waveguide.

\[ \langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_c \ N_w \ g \ W_1 \ W_e1 \ L_1 \ H \ g_2 \ W_2 \ W_e2 \ L_2 \ EC \ ringSymInvPosLCDS \rangle \]

**Figure 2.198:** Example shape illustrating various parameters from the `ringSymInvPosLCDS` constructor.
Ring symmetric inverse positive structure with an additional coupling pulley.

\[ <x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{c1} \ L_1 \ g_2 \ L_{c2} \ W_2 \ W_{c2} \ L_2 \ H_2 \ EC \ ringSymInvPosLCPul > \]

**Figure 2.199:** Example shape illustrating various parameters from the `ringSymInvPosLCPul` constructor.
### 2.9.9.10 Disc-Ring Symmetric Inverse Positive Tone Arc

Symmetric inverse positive tone structures constructed using arcs are similar to ones described in section 2.9.9.6. Here, a slot waveguide is formed at a distance \( g \) away from a disc or a ring structure using a positive resist exposure of the exposure sleeve elements \( W_e \) and \( r_e \).

\[
<x \ y \ r_d \ r_e \ N \ g \ \theta \ N_w \ W \ W_e \ L_e \ EC \ discSymmetricInvPosA>
\]
\[
<x \ y \ r_i \ W_r \ r_e \ N \ g \ \theta \ N_w \ W \ W_e \ L_e \ EC \ ringSymmetricInvPosA>
\]

![Figure 2.200: Example shapes illustrating various parameters from the (a) discSymmetricInvPosA and (b) ringSymmetricInvPosA constructors.](image-url)
Disc symmetric inverse positive arc structure with an additional coupling waveguide.

\[
<x \ y \ rd \ re \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ discSymInvPosADS>
\]

Figure 2.201: Example shapes illustrating various parameters from the (a) discSymInvPosADS and (b) ringSymInvPosADS constructors.
Disc symmetric inverse positive arc structure with an additional coupling pulley.

\[<x \ y \ r_d \ r_e \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ \theta_2 \ W_2 \ W_{e2} \ L_{S2} \ EC \ discSymInvPosAPul>\]

Figure 2.202: Example shape illustrating various parameters from the discSymInvPosAPul constructor.
Ring symmetric inverse positive arc structure with an additional coupling waveguide.

\[
\begin{align*}
<x & y & r_i & W_r & r_e & N & \theta & N_{wg} & W & W_c & L_S & g_2 & W_2 & W_{c2} & L_2 & EC & \text{ringSymInvPosADS}
\end{align*}
\]

**Figure 2.203:** Example shape illustrating various parameters from the `ringSymInvPosADS` constructor.
Ring symmetric inverse positive arc structure with an additional coupling pulley.

Figure 2.204: Example shape illustrating various parameters from the \texttt{ringSymInvPosAPul} constructor.
The following arc based, disc-ring waveguide symmetric inverse positive tone structures are defined by the coupling length parameter $L_c$.

\[
\langle x \ y \ r_d \ \ r_e \ N \ g \ L_c \ N_wg \ W \ W_e \ L_S \ EC \ discSymmetricInvPosLCA> \\
\langle x \ y \ r_i \ W_r \ r_e \ N \ g \ L_c \ N_wg \ W \ W_e \ L_S \ EC \ ringSymmetricInvPosLCA> \\
\]

Figure 2.205: Example shapes illustrating various parameters from the (a) discSymmetricInvPosLCA and (b) ringSymmetricInvPosLCA constructors.
Disc symmetric inverse positive arc structure with an additional coupling waveguide.

\[
<x\ y\ rd\ re\ N\ g_1\ L_c\ N_wg\ W_1\ W_{e1}\ L_S\ g_2\ W_2\ W_{e2}\ L_2\ EC\ discSymInvPosLCADS>
\]

Figure 2.206: Example shape illustrating various parameters from the \texttt{discSymInvPosLCADS} constructor.
Disc symmetric inverse positive arc structure with an additional coupling pulley.

\[ <x \ y \ rd \ rc \ N \ g_1 \ L_{c_1} \ N_{wg} \ W_1 \ W_{c_1} \ L_{S_1} \ g_2 \ L_{c_2} \ W_2 \ W_{c_2} \ L_{S_2} \ EC \ discSymInvPosLCAPul> \]

**Figure 2.207:** Example shape illustrating various parameters from the `discSymInvPosLCAPul` constructor.
Disc symmetric inverse positive arc structure with an additional coupling waveguide.

\[
< x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{c1} \ L_S \ g_2 \ W_2 \ W_{c2} \ L_2 \ EC \ ringSymInvPosLCADS >
\]

Figure 2.208: Example shape illustrating various parameters from the `ringSymInvPosLCADS` constructor.
Disc symmetric inverse positive arc structure with an additional coupling waveguide.

Figure 2.209: Example shape illustrating various parameters from the `ringSymInvPosLCAPul` constructor.
2.9.9.11 Disc-Ring Pulley Bezier

Shapes characterized by a curved waveguide around a disc \( R_{\text{waveGuide}} > -R_{\text{Disc}} \) and ring \( R_{\text{waveGuide}} > -R_{\text{Ring}} \) objects. Both objects (disc/ring) are described by the center coordinate \((x, y)\), number of segments \(N\), gap between the object and waveguide \(g\), waveguide opening angle \(\theta\), number of sides \(N_{\text{wg}}\), width \(W\), length \(L\) and height \(H\). Disc object is defined by a radius \(r_d\) and the ring is characterized by an inner radius \(r_i\) and ring width \(W_r\).

\[
< x \ y \ r_d \ N \ g \ \theta \ N_{\text{wg}} \ W \ L \ H \ EC \ discPulley >
\]

\[
< x \ y \ r_i \ W_r \ N \ g \ \theta \ N_{\text{wg}} \ W \ L \ H \ EC \ ringPulley >
\]

Figure 2.210: Example shapes illustrating various parameters of the (a) discPulley and (b) ringPulley constructors.
Disc pulley structure with an additional coupling waveguide.

\[
\begin{align*}
&<x \ y \ r_d \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ L_1 \ H \ g_2 \ W_2 \ L_2 \ EC \ discPulDS> \\
&\text{Figure 2.211: Example shape illustrating various parameters of the discPulDS constructor.}
\end{align*}
\]
Disc pulley structure with an additional coupling pulley.

\[
\begin{align*}
&x \quad y \quad r_d \quad N_1 \quad \theta_1 \quad N_{wg} \quad W_1 \quad L_1 \quad H_1 \quad g_2 \quad \theta_2 \quad W_2 \quad L_2 \quad H_2 \quad EC \quad \text{discPulPul}
\end{align*}
\]

Figure 2.212: Example shape illustrating various parameters of the discPulPul constructor.
Ring pulley structure with an additional coupling waveguide.

<\(x\ y\ r_i\ W_r\ N\ g_1\ \theta\ N_{wg}\ W_1\ L_1\ H\ g_2\ W_2\ L_2\ EC\ \text{ringPulDS}\)>

**Figure 2.213:** Example shape illustrating various parameters of the **ringPulDS** constructor.
Ring pulley structure with an additional coupling pulley.

Figure 2.214: Example shape illustrating various parameters of the ringPulPul constructor.
The following disc-ring waveguide pulley structures are defined by the coupling length parameter $L_c$.

\[
\begin{align*}
&\langle x \ y \ r_d \ N \ g \ L_c \ N_{wg} \ W \ L \ H \ EC \ discPulleyLC> \\
&\langle x \ y \ r_i \ W_r \ N \ g \ L_c \ N_{wg} \ W \ L \ H \ EC \ ringPulleyLC>
\end{align*}
\]

Figure 2.215: Example shapes illustrating various parameters of the (a) discPulleyLC and (b) ringPulleyLC constructors.
Disc pulley structure with an additional coupling waveguide.

\[
\begin{align*}
&<x \ y \ r_d \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ L_1 \ H \ g_2 \ W_2 \ L_2 \ EC \ discPulLCDS>
\end{align*}
\]

Figure 2.216: Example shape illustrating various parameters of the discPulLCDS constructor.
Disc pulley structure with an additional coupling pulley.

\[ <x \ y \ r_d \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_1 \ H_1 \ g_2 \ L_{c2} \ W_2 \ L_2 \ H_2 \ EC \ discPulLCPul> \]

Figure 2.217: Example shape illustrating various parameters of the discPulLCPul constructor.
Ring pulley structure with an additional coupling waveguide.

\[ <x \ y \ r_i \ W_r \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ L_1 \ H \ g_2 \ W_2 \ L_2 \ EC \ \text{ringPulLCDS}> \]

Figure 2.218: Example shape illustrating various parameters of the \text{ringPulLCDS} constructor.
Ring pulley structure with an additional coupling pulley.

\[
<x\ y\ r_i\ W_r\ N\ g_1\ L_{c1}\ N_wg\ W_1\ L_1\ H\ g_2\ L_{c2}\ W_2\ L_2\ H_2\ EC\ ringPulLCpul>
\]

Figure 2.219: Example shape illustrating various parameters of the ringPulLCpul constructor.
2.9.9.12 Disc-Ring Pulley Arc

Symmetric structures constructed using arcs are similar to ones described in section 2.9.9.11. Both objects (disc/ring) are described by the center coordinate \((x, y)\), number of segments \((N)\), gap between the object and waveguide \((g)\), waveguide opening angle \((\theta)\), number of sides \((N_{wg})\), width \((W)\), and length of the connecting straight waveguide section \((L_S)\). Disc object is defined by a radius \((r_d)\) and the ring is characterized by an inner radius \((r_i)\) and ring width \((W_r)\). \(EC = 0\) and \(EC = 1\) for waveguides without and with endcaps, respectively.

\[
<x, y, r_d, N, g, \theta, N_{wg}, W, L_S, EC \text{ discPulleyA}>
\]

\[
<x, y, r_i, W_r, N, g, \theta, N_{wg}, W, L_S, EC \text{ ringPulleyA}>
\]

Figure 2.220: Example shapes illustrating various parameters of the (a) discPulleyA and (b) ringPulleyA constructors.
Disc pulley arc structure with an additional coupling waveguide.

\[
\begin{align*}
&< x \ y \ r_d \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ discPulADS >
\end{align*}
\]

![Diagram](image)

**Figure 2.221:** Example shape illustrating various parameters of the *discPulADS* constructor.
Disc pulley arc structure with an additional coupling pulley.

\[ <x \ y \ r_d \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ \theta_2 \ W_2 \ L_{S2} \ EC \ discPulAPul> \]

Figure 2.222: Example shape illustrating various parameters of the discPulAPul constructor.
Ring pulley arc structure with an additional coupling waveguide.

\[<x \quad y \quad r_i \quad W_r \quad N \quad g_1 \quad \theta \quad N_{wg} \quad W_1 \quad L_S \quad g_2 \quad W_2 \quad L_2 \quad EC \quad \text{ringPulADS}>\]

Figure 2.223: Example shape illustrating various parameters of the ringPulADS constructor.
Ring pulley arc structure with an additional coupling pulley.

Figure 2.224: Example shape illustrating various parameters of the ringPulAPul constructor.
The following arc based, disc-ring waveguide pulley structures are defined by the coupling length parameter $L_c$.

$$<x\ y\ r_d\ N\ g\ L_c\ N_{wg}\ W\ L_S\ EC\ \text{discPulleyLCA}>$$

$$<x\ y\ r_i\ W_r\ N\ g\ L_c\ N_{wg}\ W\ L_S\ EC\ \text{ringPulleyLCA}>$$

![Diagram of pulley structures](image)

**Figure 2.225:** Example shapes illustrating various parameters of the (a) *discPulleyLCA* and (b) *ringPulleyLCA* constructors.
Disc pulley arc structure with an additional coupling waveguide.

\[ \{x, y, r_d, N, g_1, L_c, N_{wg}, W_1, L_S, g_2, W_2, L_2, EC, \text{discPulLCADS}\} \]

Figure 2.226: Example shape illustrating various parameters of the (a) \text{discPulLCADS} constructor.
Disc pulley arc structure with an additional coupling pulley.

\[
<x, y, r_d, N, g_1, L_{c1}, N_{wg}, W_1, L_{S1}, g_2, L_{c1}, W_2, L_{S2}, EC, \text{discPulLCAPul}>
\]

Figure 2.227: Example shape illustrating various parameters of the (a) discPulLCAPul constructor.
Ring pulley arc structure with an additional coupling waveguide.

\[ <x \ y \ r_i \ W_r \ g_1 \ L_c \ N_{wg} \ W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ \text{ringPullCADS}> \]

**Figure 2.228:** Example shape illustrating various parameters of the `ringPullCADS` constructor.
Ring pulley arc structure with an additional coupling pulley.

\[
<x\ y\ r_i\ W_r\ N\ g_1\ L_{c1}\ N_{wg}\ W_1\ L_{S1}\ g_2\ L_{c2}\ W_2\ L_{S2}\ \text{EC}\ \text{ringPulLCAPul}>
\]

Figure 2.229: Example shape illustrating various parameters of the \text{ringPulLCAPul} constructor.
2.9.9.13 Disc-Ring Pulley Inverse Bezier

The structure is similar to the disc-ring waveguide symmetric structure defined in the previous section (2.9.9.11). Here, a slot waveguide of width $W$ is formed by exposing a surrounding rectangular region of width $W_e$. Using a negative tone resist, the resulting structure would produce a disc or ring shape at a distance $g$ to a slotted waveguide.

$\text{discPulleyInv}$

$\text{ringPulleyInv}$

Figure 2.230: Example shapes illustrating various parameters of the (a) $\text{discPulleyInv}$ and (b) $\text{ringPulleyInv}$ constructors.
Disc pulley inverse structure with an additional coupling waveguide.

\[ <x, y, r_d, N, g_2, \theta, N_{wg}, W_2, W_{c2}, L_2, H, g_2, W_2, L_2, EC, \text{discPulInvDS}> \]

Figure 2.231: Example shape illustrating various parameters of the discPulInvDS constructor.
Disc pulley inverse structure with an additional coupling pulley.

\[ <x \ y \ r_d \ N \ g_2 \ \theta_1 \ N_{wg} \ W_2 \ W_{e2} \ L_2 \ H_2 \ g_2 \ \theta_2 \ W_2 \ W_{e2} \ L_2 \ H_2 \ EC \ discPullInvPul> \]

**Figure 2.232:** Example shape illustrating various parameters of the discPullInvPul constructor.
Ring pulley inverse structure with an additional coupling waveguide.

\[ (x, y, r_i, W_r, N, g_1, \theta, N_{wg}, W_1, W_{e1}, L_1, H, g_2, W_2, W_{e2}, L_2, EC) \text{ ringPullInvDS} \]

Figure 2.233: Example shape illustrating various parameters of the \texttt{ringPullInvDS} constructor.
Ring pulley inverse structure with an additional coupling pulley.

\[
\langle x \ y \ r_i \ W_r \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ W_{c1} \ L_1 \ H \ g_2 \ \theta_2 \ W_2 \ W_{c2} \ L_2 \ H_2 \ EC \ ringPulInvPul\rangle
\]

Figure 2.234: Example shape illustrating various parameters of the ringPulInvPul constructor.
The following disc-ring waveguide pulley inverse structures are defined by the coupling length parameter $L_c$.

\[ <x\ y\ r_d\ N\ g\ L_c\ N_wg\ W\ W_t\ L\ H\ EC\ \text{discPulleyInvLC}> \]

\[ <x\ y\ r_i\ W_t\ N\ g\ L_c\ N_wg\ W\ W_t\ L\ H\ EC\ \text{ringPulleyInvLC}> \]

**Figure 2.235:** Example shapes illustrating various parameters of the (a) discPulleyInvLC and (b) ringPulleyInvLC constructors.
Disc pulley inverse structure with an additional coupling waveguide.

\[ <x\ y\ r_d\ N\ g_1\ L_c\ N_{wg}\ W_1\ W_{c1}\ L_1\ H\ g_2\ W_2\ W_{c2}\ L_2\ EC\ discPulInvLCDS> \]

**Figure 2.236:** Example shape illustrating various parameters of the `discPulInvLCDS` constructor.
Disc pulley inverse structure with an additional coupling pulley.

\[ <x \ y \ r_d \ N \ g_1 \ L_{e1} \ N_{w1} \ W_1 \ W_{c1} \ L_1 \ H_2 \ g_2 \ L_{c2} \ W_2 \ W_{c2} \ L_2 \ H_2 \ EC \ discPulInvLCPul> \]

Figure 2.237: Example shape illustrating various parameters of the discPulInvLCPul constructor.
Ring pulley inverse structure with an additional coupling waveguide.

$$\langle x \ y \ r_i \ W_r \ N_g \ W_{c_1} \ W_{c_2} \ L_1 \ H \ g_2 \ W_1 \ L_{c} \ N_{w_g} \ W_2 \ L_2 \ EC \ ringPullInvLCDS \rangle$$

Figure 2.238: Example shape illustrating various parameters of the `ringPullinvLCDS` constructor.
Ring pulley inverse structure with an additional coupling pulley.

\[ <x \ y \ r_i \ W_r \ N_{g1} \ L_{c1} \ N_{wg1} \ W_1 \ W_{c1} \ L_1 \ H_1 \ g_2 \ L_{c2} \ W_2 \ W_{c2} \ L_2 \ H_2 \ EC > \]

Figure 2.239: Example shape illustrating various parameters of the \texttt{ringPullInvLCPul} constructor.
2.9.9.14 Disc-Ring Pulley Inverse Arc

Pulley inverse structures constructed using arcs are similar to ones described in section 2.9.9.12. Here, a slot waveguide of width $W$ is formed by exposing a surrounding rectangular region of width $W_e$. Using a negative tone resist, the resulting structure would produce a disc or ring shape at a distance $g$ to a slotted waveguide.

$\langle x \ y \ r_d \ N \ g \ \theta \ N_{wg} \ W \ W_e \ L_S \ EC \ discPulleyInvA \rangle$

$\langle x \ y \ r_i \ W_i \ N \ g \ \theta \ N_{wg} \ W \ W_e \ L_S \ EC \ ringPulleyInvA \rangle$

Figure 2.240: Example shapes illustrating various parameters of the (a) discPulleyInvA and (b) ringPulleyInvA constructors.
Disc pulley inverse arc structure with an additional coupling waveguide.

\[
\begin{align*}
\text{x} & \quad \text{y} & \quad r_d & \quad N & \quad g_1 & \quad \theta & \quad N_{wg} & \quad W_1 & \quad W_{e1} & \quad L_S & \quad g_2 & \quad W_2 & \quad W_{e2} & \quad L_2 & \quad EC & \quad \text{discPullInvADS}
\end{align*}
\]

Figure 2.241: Example shape illustrating various parameters of the discPullInvADS constructor.
Disc pulley inverse arc structure with an additional coupling pulley.

\[ <x \ y \ rd \ N_1 \ \theta_1 \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ \theta_2 \ W_2 \ L_{S2} \ EC \ discPulInvAPul> \]

**Figure 2.242:** Example shape illustrating various parameters of the `discPulInvAPul` constructor.
Ring pulley inverse arc structure with an additional coupling waveguide.

$\langle x, y, r_{i}, W_{r}, N_{g_{1}}, \theta, N_{w_{3}}, W_{1}, W_{c_{1}}, L_{S}, g_{2}, W_{2}, W_{c_{2}}, L_{2}, EC \rangle$ ringPullInvADS

Figure 2.243: Example shape illustrating various parameters of the ringPullInvADS constructor.
Ring pulley inverse arc structure with an additional coupling pulley.

\[ <x \ y \ r_i \ W_r \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ \theta_2 \ W_2 \ W_{e2} \ L_2 \ EC \ ringPulInvAPul> \]

Figure 2.244: Example shape illustrating various parameters of the \texttt{ringPulInvAPul} constructor.
The following arc based, disc-ring waveguide pulley inverse structures are defined by the coupling length parameter $L_c$.

$$<x\ y\ r_d\ N\ g\ L_c\ N_{wg}\ W_e\ L_S\ EC\ \text{discPulleyInvLCA}>$$
$$<x\ y\ r_i\ W_r\ N\ g\ L_c\ N_{wg}\ W_e\ L_S\ EC\ \text{ringPulleyInvLCA}>$$

Figure 2.245: Example shapes illustrating various parameters of the (a) discPulleyInvLCA and (b) ringPulleyInvLCA constructors.
Disc pulley inverse arc structure with an additional coupling waveguide.

\[
\begin{align*}
\langle x & y \ r_d \ N_1 \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ discPulInvLCADS \rangle
\end{align*}
\]

**Figure 2.246:** Example shape illustrating various parameters of the *discPulInvLCADS* constructor.
Disc pulley inverse arc structure with an additional coupling pulley.

\[ <x \ y \ rd \ N \ g_1 \ L_c1 \ N_wg \ W_1 \ W_c1 \ L_S1 \ g_2 \ L_c2 \ W_2 \ W_c2 \ L_S2 \ EC \ discPulInvLCAPul> \]

**Figure 2.247:** Example shape illustrating various parameters of the `discPulInvLCAPul` constructor.
Ring pulley inverse arc structure with an additional coupling waveguide.

\[
\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{c1} \ L_S \ g_2 \ W_2 \ W_{c2} \ L_2 \ EC \ \text{ringPulInvLCADS}\rangle
\]

Figure 2.248: Example shape illustrating various parameters of the \textit{ringPulInvLCADS} constructor.
Ring pulley inverse arc structure with an additional coupling pulley.

\[
\begin{align*}
&\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_{c_1} \ N_{wg} \ W_1 \ W_{c_1} \ L_{S_1} \ g_2 \ L_{c_2} \ W_2 \ W_{c_2} \ L_{S_2} \ EC \ ringPulInvLCA\text{Pul}\rangle 
\end{align*}
\]

Figure 2.249: Example shape illustrating various parameters of the ringPulInvLCA\text{Pul} constructor.
2.9.9.15 Disc-Ring Pulley Inverse Positive Tone Bezier

The structure is similar to the disc-ring waveguide symmetric structure defined in section 2.9.9.11. Here, a slot waveguide is formed at a distance \( g \) away from a disc or a ring structure using a positive resist exposure of the exposure sleeve elements \( W_e \) and \( r_e \).

\[
\begin{align*}
&\langle x, y, r_d, r_e, N, g, \theta, N_{wg}, W_1, W_e, L_1, H, EC \rangle \text{ discPulleyInvPos} \\
&\langle x, y, r_i, W_r, r_e, N, g, \theta, N_{wg}, W_1, W_e, L_1, H, EC \rangle \text{ ringPulleyInvPos}
\end{align*}
\]

Figure 2.250: Example shapes illustrating various parameters of the (a) discPulleyInvPos and (b) ringPulleyInvPos constructors.
Disc pulley inverse positive structure with an additional coupling waveguide.

\[
\begin{aligned}
&<x \ y \ r_d \ r_e \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ W_{e1} \ L_1 \ H \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ discPulInvPDS>
\end{aligned}
\]

Figure 2.251: Example shape illustrating various parameters of the discPulInvPDS constructor.
Disc pulley inverse positive structure with an additional coupling pulley.

\[
<x \ y \ r_d \ r_e \ N \ g_1 \ \theta_1 \ N_{wg} \ W_1 \ W_{e1} \ L_1 \ H \ g_2 \ \theta_2 \ W_2 \ W_{e2} \ L_2 \ H_2 \ EC \ discPulInvPPul>
\]

Figure 2.252: Example shape illustrating various parameters of the discPulInvPPul constructor.
Ring pulley inverse positive structure with an additional coupling waveguide.

Figure 2.253: Example shape illustrating various parameters of the ringPullInvPDS constructor.
Ring pulley inverse positive structure with an additional coupling pulley.

\[ \langle x \ y \ r_i \ W_r \ r_c \ N \ g_1 \ \theta_1 \ N_w \ g \ W_1 \ W_{c1} \ L_1 \ H_1 \ g_2 \ \theta_2 \ W_2 \ W_{c2} \ L_2 \ H_2 \ EC \ \text{ringPulInvPPul} \rangle \]

Figure 2.254: Example shape illustrating various parameters of the \text{ringPulInvPPul} constructor.
The following disc-ring waveguide pulley inverse positive tone structures are defined by the coupling length parameter $L_c$.

$$\langle x \ y \ r_d \ r_e \ N \ g \ L_c \ N_wg \ W \ W_e \ L \ H \ EC \ discPulleyInvPosLC \rangle$$

$$\langle x \ y \ r_i \ W_r \ r_e \ N \ g \ L_c \ N_wg \ W \ W_e \ L \ H \ EC \ ringPulleyInvPosLC \rangle$$

![Diagram of disc-ring waveguide pulley inverse positive tone structures](image)

**Figure 2.255:** Example shapes illustrating various parameters of the (a) discPulleyInvPosLC and (b) ringPulleyInvPosLC constructors.
Disc pulley inverse positive structure with an additional coupling waveguide.

\[ <x \ y \ r_d \ r_e \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_1 \ H \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ discPulInvPLCDS> \]

Figure 2.256: Example shape illustrating various parameters of the \textit{discPulInvPLCDS} constructor.
Disc pulley inverse positive structure with an additional coupling pulley.

\[
<x y r_d r_c N g_1 L_c N_{wg} W_1 W_{c1} L_1 H_1 g_2 L_{c2} W_2 W_{c2} L_2 H_2 EC \text{ discPulInvPLCPul}>
\]

Figure 2.257: Example shape illustrating various parameters of the \text{discPulInvPLCPul} constructor.
Disc pulley inverse positive structure with an additional coupling waveguide.

\[ <x \ y \ r_i \ W_r \ r_c \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{c1} \ L_1 \ H \ g_2 \ W_2 \ W_{c2} \ L_2 \ EC \ \text{ringPullInvPLCDS}> \]

Figure 2.258: Example shape illustrating various parameters of the \text{ringPullInvPLCDS} constructor.
Disc pulley inverse positive structure with an additional coupling pulley.

\[ <x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_{c_1} \ N_{wg} \ W_1 \ W_{e_1} \ L_1 \ g_2 \ L_{c_2} \ W_2 \ W_{e_2} \ L_2 \ H_2 \ EC \ \text{ringPullInvPLCPul}> \]

Figure 2.259: Example shape illustrating various parameters of the \text{ringPullInvPLCPul} constructor.
2.9.9.16 Disc-Ring Pulley Inverse Positive Tone Arc

Pulley inverse positive tone structures constructed using arcs are similar to ones described in section 2.9.9.12. Here, a slot waveguide is formed at a distance $g$ away from a disc or a ring structure using a positive resist exposure of the exposure sleeve elements $W_e$ and $r_e$.

$$<x\ y\ r_d\ r_e\ N\ g\ \theta\ N_{wg}\ W\ W_e\ L_S\ EC\ discPulleyInvPosA>$$

$$<x\ y\ r_i\ W_i\ r_e\ N\ g\ \theta\ N_{wg}\ W\ W_e\ L_S\ EC\ ringPulleyInvPosA>$$

Figure 2.260: Example shapes illustrating various parameters of the (a) discPulleyInvPosA and (b) ringPulleyInvPosA constructors.
Disc pulley inverse positive arc structure with an additional coupling waveguide.

\[ <x \ y \ r_d \ r_e \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ discPullInvPADS> \]

**Figure 2.261:** Example shape illustrating various parameters of the *discPullInvPADS* constructor.
Disc pulley inverse positive arc structure with an additional coupling pulley.

\[ <x \ y \ rd \ \theta_1 \ \theta_2 \ N_g1 \ N_{wg} \ W_1 \ W_c1 \ L_{S1} \ g_2 \ W_2 \ W_c2 \ L_{S2} \ EC \ discPulInvPAPul> \]

**Figure 2.262:** Example shape illustrating various parameters of the `discPulInvPAPul` constructor.
Ring pulley inverse positive arc structure with an additional coupling waveguide.

\[
\langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ \theta \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringPullInvPADS}\rangle
\]

**Figure 2.263:** Example shape illustrating various parameters of the `ringPullInvPADS` constructor.
Ring pulley inverse positive arc structure with an additional coupling pulley.

\[
\begin{align*}
&< x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ \theta_1 \ N_w \ g_1 \ \theta_1 \ W_1 \ L_{S1} \ g_2 \ \theta_2 \ W_2 \ L_2 \ EC \ \text{ringPullInvPAPul}> \\
&
\end{align*}
\]

Figure 2.264: Example shape illustrating various parameters of the \textit{ringPullInvPAPul} constructor.
The following disc-ring waveguide pulley inverse positive tone structures are defined by the coupling length parameter $L_c$.

$$\langle x \ y \ r_d \ r_e \ N \ g \ L_c \ N_{wg} \ W \ W_e \ L_S \ EC \ discPulleyInvPosLCA\rangle$$

$$\langle x \ y \ r_i \ W_r \ r_e \ N \ g \ L_c \ N_{wg} \ W \ W_e \ L_S \ EC \ ringPulleyInvPosLCA\rangle$$

Figure 2.265: Example shapes illustrating various parameters of the (a) discPulleyInvPosLCA and (b) ringPulleyInvPosLCA constructors.
Disc pulley inverse positive arc structure with an additional coupling waveguide.

\[<x \; y \; r_d \; r_e \; N \; g_1 \; L_c \; N_wg \; W_1 \; W_{e1} \; L_S \; g_2 \; W_2 \; W_{e2} \; L_2 \; EC \; discPullInvPLCADS>\]

Figure 2.266: Example shape illustrating various parameters of the `discPullInvPLCADS` constructor.
Disc pulley inverse positive arc structure with an additional coupling pulley.

\[ \langle x \ y \ r_d \ r_e \ N \ g_1 \ L_{c_1} \ N_{w_1} \ W_1 \ W_{c_1} \ L_{S_1} \ g_2 \ L_{c_2} \ W_2 \ W_{c_2} \ L_2 \ EC \ discPulInvPULCAPul \rangle \]

Figure 2.267: Example shape illustrating various parameters of the discPulInvPULCAPul constructor.
Ring pulley inverse positive arc structure with an additional coupling waveguide.

\[ <x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ ringPulInvPLCADS> \]

Figure 2.268: Example shape illustrating various parameters of the `ringPulInvPLCADS` constructor.
Ring pulley inverse positive arc structure with an additional coupling pulley.

\( <x \ y \ r_i \ W_i \ r_e \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_{c2} \ W_2 \ L_{c2} \ EC \ ringPulInvPLCA\text{Pul}> \)

Figure 2.269: Example shape illustrating various parameters of the \textit{ringPulInvPLCA\text{Pul}} constructor.
2.9.10 Waveguide Inverse Photonic Crystals (Ellipse)

Waveguide shapes characterized by elliptical perforations on either a periodic ($a = constant$) or varying grid ($a_1, a_2, ..., a_n$). Elliptical structures are defined by the two radii ($r_x$ and $r_y$), the number of shape sides ($N$) and the waveguide width ($W$). The center waveguide position is $(x, y)$. Indexed numbers (1 – 7 and 1 – 5) correspond to the ellipse directly below.

\[ r_{x1} \ r_{y1} \ N_1 \ r_{x2} \ r_{y2} \ N_2 \ ... \ r_{xn} \ r_{yn} \ N_n \ x \ y \ a \ W \ \text{waveGuideInvPhC} \]

\[ r_{x1} \ r_{y1} \ a_1 \ N_1 \ r_{x2} \ r_{y2} \ a_2 \ N_2 \ ... \ r_{xn} \ r_{yn} \ a_n \ N_n \ x \ y \ W \ \text{waveGuideInvPhCvary} \]

![Diagram of elliptical waveguide shapes](image)

**Figure 2.270:** Example shapes illustrating various parameters from the (a) waveGuideInvPhC and (b) waveGuideInvPhCvary constructor.
2.9.11 Waveguide Photonic Crystals (Ellipse)

Waveguide shapes characterized by elliptical structures on either a periodic \((a = \text{constant})\) or varying grid \((a_1, a_2, \ldots a_n)\). Elliptical structures are defined by the two radii \((r_x \text{ and } r_y)\), the number of shape sides \((N)\) and the waveguide width \((W)\). The center waveguide position is \((x, y)\). The box defining the waveguide extends beyond the elliptical structures by the overhang \((S_o)\) and is of height \((S_h)\). Indexed numbers \((1−7 \text{ and } 1−5)\) correspond to the ellipse directly below.

\[
\begin{align*}
&\text{waveGuidePhC} \\
&\text{waveGuidePhCvary}
\end{align*}
\]

**Figure 2.271:** Example shapes illustrating various parameters from the (a) \text{waveGuidePhC} and (b) \text{waveGuidePhCvary} constructor.
2.9.12 Waveguide Inverse Photonic Crystals (Rectangle)

Waveguide shapes characterized by rectangular perforations on either a periodic \((a = \text{constant})\) or varying grid \((a_1, a_2, \ldots, a_n)\). Rectangular structures are defined by the \(L_x\) and \(L_y\), and the waveguide width \((W)\). The center waveguide position is \((x, y)\). Indexed numbers \((1 – 7\) and \(1 – 5\)) correspond to the rectangles directly below.

\[
L_{x_1} L_{y_1} L_{x_2} L_{y_2} \ldots L_{x_n} L_{y_n} x \ y \ a \ W \ \text{waveGuideInvRectPhC}
\]

\[
L_{x_1} L_{y_1} a_1 L_{x_2} L_{y_2} a_2 \ldots L_{x_n} L_{y_n} a_n x y W \ \text{waveGuideInvRectPhCvary}
\]

Figure 2.272: Example shapes illustrating various parameters from the (a) \text{waveGuideInvRectPhC} and (b) \text{waveGuideInvRectPhCvary} constructor.
Waveguide shapes characterized by rectangular structures on either a periodic \((a_1 = \text{constant})\) or varying grid \((a_1, a_2, \ldots, a_n)\). Rectangular structures are defined by the waveguide position \((x, y)\) and the waveguide width \((W)\). The center waveguide position \((x, y)\) is \((L_x, L_y)\) and the waveguide extends beyond the rectangular structures by the overhang \((S_o)\) and is of height \((S_h)\). Indexed numbers \((1 \ldots 7)\) correspond to the rectangles directly below.

Figure 2.273: Example shapes illustrating various parameters from the (a) \texttt{waveGuideRectPhC} and (b) \texttt{waveGuideRectPhC\_vary} constructor.
2.9.14 Waveguide Photonic Crystals (Flush Rectangle)

Waveguide shapes characterized by rectangular structures on a varying grid \((a_1, a_2, \ldots, a_n)\). Rectangular structures are defined by the \(L_x\) and \(L_y\) positioned flush at a distance \(d\) away from top of waveguide, and the waveguide width \((W)\). The center waveguide position is \((x, y)\). The box defining the waveguide (Figure 2.9.14) extends beyond the rectangular structures by the overhang \((S_o)\) and is of height \((S_h)\). Indexed numbers \((1 - 5)\) correspond to the rectangles directly below.

\[
\begin{align*}
L_{x_1} & \quad L_{y_1} & \quad a_1 & \quad L_{x_2} & \quad L_{y_2} & \quad a_2 & \quad \ldots & \quad L_{x_n} & \quad L_{y_n} & \quad a_n & \quad x \quad y \quad W \quad d & \quad \text{waveGuideInvRectFlushPhCvary} \\
L_{x_1} & \quad L_{y_1} & \quad a_1 & \quad L_{x_2} & \quad L_{y_2} & \quad a_2 & \quad \ldots & \quad L_{x_n} & \quad L_{y_n} & \quad a_n & \quad x \quad y \quad W \quad d \quad S_o \quad S_h & \quad \text{waveGuideRectFlushPhCvary}
\end{align*}
\]

![Diagram of waveguide shapes](image)

**Figure 2.274:** Example shapes illustrating various parameters from the (a) *waveGuideInvRectFlushPhCvary* and (b) *waveGuideRectFlushPhCvary* constructor.
2.9.15 Various Waveguide-Disc-Tip Coupled Structures

\[ <x \ y \ r \ N_{sides} \ d_x \ d_y \ w \ L_{wg} \ r_{wg} \ s \ g \ r_e \ d_c \ c \ d_c \ \theta_{(x,y)} \ \text{wgdcV1} > \]

Figure 2.275: \textit{wgdcV1} constructor.
Figure 2.276: \textit{wgdcdv2} constructor.
<x y r N_{sides} d_x d_y d_g H_1 W_1 w L_{wg} r_{wg} s g c d_c D \theta_{(x,y)} \textit{wgdcV3}>

Figure 2.277: \textit{wgdcV3} constructor.
Figure 2.278: \textit{wgcdV4} constructor.
Figure 2.279: \textit{wgdcV5} constructor.
Figure 2.280: \textit{wgdcDV6} constructor.
Figure 2.281: \textit{wgdcV7} constructor.
\[ <x \ y \ t \ b \ f \ s \ w_1 \ w_2 \ L_s \ L_t \ g \ c \ d_c \ W \ H \ L_1 \ L_2 \ L_3 \ \theta(x,y) \ \text{wdcdV8}> \]

Figure 2.282: \textit{wdcdV8} constructor.
2.10 MEMS - NEMS Library

2.10.1 Actuators

2.10.1.1 Bent Beams

Bent beams form a class of linear displacement actuators. In this case, motion occurs via thermal expansion of the beam.

\[ x \ y \ w \ l_1 \ l_2 \ l_3 \ b_H \ b_W \ a \ L_a \ \theta_{(x,y)} \text{ bentBeam} \]

Figure 2.283: Bent beam of width \( w \) suspended between two anchored electrodes. The anchor overlap and GDS layer parameters are defined by \( a \) and \( L_a \).

\[ x \ y \ w \ l_1 \ l_2 \ l_3 \ l_4 \ h \ p \ N \ c_W \ d_H \ d_W \ L_d \ b_W \ a \ L_a \ \theta_{(x,y)} \text{ bentBeamArray} \]

Figure 2.284: Array of \( N \) bent beam elements with a central rectangular structure. The \( d_H \) and \( d_W \) parameter defines an extra GDS layer \( (L_d) \) that could be used to perforate the central structure.
2.10.1.2 Bi-Morph Thermal Actuator

Differential thermal expansion of the bimorph actuator allows for in-plane motion. This occurs when a voltage is applied, wherein current induced Joule heating causes uneven thermal expansion within the two beams of differing widths.

\[
x\ y\ \ w_1\ w_2\ w_3\ w_4\ l_1\ l_2\ l_3\ p\ s_H\ s_W\ L_s\ b_H\ b_W\ a\ L_a\ \theta_{(x,y)}\ \text{biMorph}
\]

Figure 2.285: Bimorph thermal actuator.
2.10.1.3 Combs and Drive Elements

Linear comb drive created using three GDS layers. Lower and upper base electrodes are created using the initialized and $L_b$ GDS layers respectively. The anchor overlap and GDS layers are correspondingly $a$ and $L_a$.

\[
(x, y, w_1, w_2, l_1, l_2, N, p, b_H, L_b, a, L_a, \theta_{(x,y)}) \quad \text{combDriveV1}
\]

Figure 2.286: Linear comb drive.
Number of electrode elements ($N$) is used to calculate the number of rectangular rotor elements ($l_1 \times w_1$). The structure is composed from the initialized GDS layer (top and bottom electrodes), $L_a$ (anchor) and $L_b$ (rotor).

$$x \ y \ w_1 \ l_1 \ l_2 \ l_3 \ g \ N \ p \ b_H \ b_W \ p_b \ L_b \ a \ L_a \ \theta_{(x,y)} \ \text{linearDriveL1}$$

Figure 2.287: Linear drive composed of anchored electrodes and a central rotor.
2.10.1.4 Folded Springs

\[ x \quad y \quad w \quad l_1 \quad l_2 \quad p \quad A \quad N \quad b_H \quad b_W \quad a \quad L_a \quad \theta_{(x,y)} \]

\[ x \quad y \quad w \quad l_1 \quad l_2 \quad p \quad A \quad N \quad b_H \quad b_W \quad a \quad L_a \quad \theta_{(x,y)} \]

Figure 2.288: Folded springs with a single anchored pad. The spring meander initiates at the (a) midpoint and (b) bottom of pads.
Figure 2.289: Folded springs with two anchored pads. The spring meander initiates at the (a) midpoint and (b) bottom of pads.
Figure 2.290: Folded springs with two anchored pads V2C.
Figure 2.291: Folded springs with two anchored pads V2D.
Figure 2.292: Folded springs with two anchored pads V2E and V2F.
Figure 2.293: Folded springs with two anchored pads V2G.
Figure 2.294: Folded springs with two anchored pads V2H.
Figure 2.295: Folded springs with two anchored pads V2l.
Figure 2.296: Folded springs with two anchored pads V2J.
2.10.2 Bolometers

Below L-Shaped bolometer is constructed from GDS layers \( L_a, L_b, L_c, L_d, L_e, \) and \( L_f \) using the below specified parameters. Rounded rectangles (layers \( L_d \) and \( L_e \)) are defined by a radius \( r_i \). The number of points defining the curved region of the rounded rectangle is defined by the \texttt{shapeReso} parameter. Meandering curved region defined by radius \( r \) is constructed using \( N_{sides} \) number of points.

\[
x \ y \ w_1 \ w_2 \ w_3 \ g_1 \ g_2 \ g_3 \ L_1 \ L_2 \ r \ r_i \ N_{sides} \ a \ b \ c \ d \ e \ f \ L_a \ L_b \ L_c \ L_d \ L_e \ L_f \ \theta(x,y) \ \texttt{bolometerL}
\]

![Figure 2.297: L-shaped bolometer structure.](image-url)
Figure 2.298: U-shaped bolometer structure.
2.10.3 Gears

MEMS gear elements defined by the \((x, y)\) position, radius \(r\) with \(N\) sides, gear width and height, \(w\) and \(h\), respectively. Number of gear elements \(N_G\) are evenly distributed along the disc circumference.

\[
\begin{align*}
\text{gear} & : x \ y \ r \ h \ N_G \ N \ \theta_{(x,y)} \\
\text{gearT} & : x \ y \ r \ h \ N_G \ w_t \ N \ \theta_{(x,y)}
\end{align*}
\]

![Diagram of MEMS gear elements created using (a) gear and (b) gearT constructors.](image)

**Figure 2.299:** MEMS gear elements created using (a) gear and (b) gearT constructors.
2.10.3.1 Hub With Straight and Circular Springs

\[ x \ y \ w \ r_i \ w_r \ R \ \text{N}_{\text{sides}} \ \text{N}_{\text{beams}} \ \alpha \ \text{L}_{\alpha} \ \theta_{(x,y)} \ \text{straightSpring} \]

Figure 2.300: Concentric hub with straight springs.

\[ x \ y \ w \ r_i \ w_r \ R \ \text{N}_{\text{sides}} \ \text{N}_{\text{beams}} \ \alpha \ \text{L}_{\alpha} \ \theta_{(x,y)} \ \text{circularSpring} \]

Figure 2.301: Concentric hub with circular springs.
In figure 2.302, size of the surrounding electrodes depends on the number of electrodes ($N_e$) and the dimensionless parameter $\beta$, where $0 < \beta < 1$. The gap between the electrodes is defined as $\beta L_{es}$, where the electrode length segment is defined as $L_{es} = C/N_e$. $C = 2\pi r^*$ is the circumference of the innermost electrode, where $r^* = R + w_r/2 + g$. Therefore, $\beta$ defines the electrode gap as the fraction of the electrode segment. Number of sides $N_s$ parameter is used to construct all circular objects.

Figure 2.302: Concentric hub with straight springs and surrounding electrodes.
Figure 2.303: Concentric hub with circular springs and surrounding electrodes.
2.10.3.2 Radial Comb Drive

Below constructors create radial comb-drives defined by an opening angle ($\theta$), overlap angle ($\theta_o$), electrode widths ($w_1, w_2$), comb width ($w_c$) and gap ($g$), number of combs ($N_{combs}$), number of sides for each comb arc segment ($N_{sides}$), anchor overlap ($a$) and anchor GDS layer ($L_a$).

$x \ y \ w_1 \ r_1 \ w_2 \ r_2 \ w_c \ g \ N_{combs} \ N_{sides} \ \theta \ \theta_o \ a \ L_a \ \theta_{(x,y)}$ **combRadialV1**

![Radial comb drive V1](image)

*Figure 2.304: Radial comb drive V1.*
Figure 2.305: Radial comb drive V2.
2.10.4 Anchored Flexures

The following constructors create accelerometer type anchored flexures with a proof mass. In all of the cases, the rectangular base support anchor is defined by the anchor support width \( a \) and GDS layer \( L_a \). Structural layer is defined by the active layer.

Anchored Flexure V2A

\[
\begin{align*}
(x, y, w, l_1, l_2, w_m, l_m, b_H, b_W, a, L_a, \theta_{(x,y)}) & \quad \text{flexure2A}
\end{align*}
\]

![Figure 2.306: Anchored Flexure V2A with two hinges supporting a proof mass.](image)
Anchored Flexure V2B

\[ x \quad y \quad w \quad l_1 \quad l_2 \quad w_m \quad l_m \quad c_H \quad c_W \quad s_H \quad s_W \quad b_H \quad b_W \quad a \quad L_a \quad \theta_{(x,y)} \] flexure2B

Figure 2.307: Anchored Flexure V2B with two hinges supporting a proof mass.
Anchored Flexure V2C

\[ \begin{align*}
\begin{array}{llllllllllll}
x & y & w & l_1 & l_2 & l_3 & w_m & l_m & A & N_p & b_H & b_W & a & L_a & \theta_{(x,y)}
\end{array}
\end{align*} \]

flexure2C

Figure 2.308: Anchored Flexure V2C with a meander supporting a proof mass.
Figure 2.309: Anchored Flexure V2D.
Anchored Flexure V2E

Figure 2.310: Anchored Flexure V2E.
Anchored Flexure V4A

\[ (x, y, w, l_1, l_2, w_m, l_m, b_H, b_c, a, L_a, \theta_{(x,y)} \)  \textbf{flexure4A} \]

Figure 2.311: Anchored Flexure V4A with four hinges supporting a proof mass.
Anchored Flexure V4B

\[ x \ y \ w \ l_1 \ l_2 \ w_m \ g \ b_H \ b_W \ a \ L_a \ \theta_{(x,y)} \]

Figure 2.312: Anchored Flexure V4B with four hinges supporting a proof mass.
Figure 2.313: Anchored Flexure V4C with four hinges supporting a proof mass.
Anchored Flexure V4D

$x \ y \ w_1 \ w_2 \ l_1 \ l_2 \ w_m \ l_m \ g \ b_H \ b_W \ a \ L_a \ \theta_{(x,y)} \ \text{flexure4D}$

Figure 2.314: Anchored Flexure V4D with four hinges supporting a proof mass.
Anchored Flexure V4E

$\begin{align*}
& x \ y \ w_1 \ w_2 \ w_3 \ w_4 \ l_1 \ l_2 \ l_3 \ l_4 \ l_5 \ b_H \ b_W \ a \ L_d \ \theta_{(x,y)} \\
& \text{flexure4E}
\end{align*}$

Figure 2.315: Anchored Flexure V4E.
2.10.5 Cantilevers

The following cantilever beam structures have a linear, percentage or sinusoidal length variation with respect to the starting length ($s_L$). The beams are periodically arranged on a base of height $b_H$ and of width that is a function of the number of elements ($n$), beam width ($w$), pitch ($p$) and base extent ($b_e$).

\[
x, y, w, s_L, p, n, b_H, b_e, \Delta L, \theta_{(x,y)} \text{ cantileverL} \]
\[
x, y, w, s_L, p, n, b_H, b_e, \text{ Percent}, \theta_{(x,y)} \text{ cantileverP} \]
\[
x, y, w, s_L, p, n, b_H, b_e, \text{ Amplitude}, \theta_{(x,y)} \text{ cantileverSine} \]

Figure 2.316: Cantilever arrays of varying length. (a) Linear (cantileverL) variation from $s_L$ with $\Delta L$ increments (b) Percentage (cantileverP) variation starting from $s_L$ and (c) sinusoidal (cantileverSine) variation over one period with amplitude $A$. 
Similar to figure 2.316, the cantilever variation for the below constructors is bound between the starting length ($s_L$) and the end length ($e_L$). Here, the origin $(x, y)$ is positioned at the lower left corner of the base, $w, s_L, e_L$, and $p$ are the cantilever width, start length, end lengths, and pitch, respectively. $n$ is the number of cantilever beams, $b_e$ and $b_H$ are the base extent and height. $\theta_{(x,y)}$ is the rotation about the origin $(x, y)$.

\[
x \ y \ w \ s_L \ e_L \ p \ n \ b_H \ b_e \ \theta_{(x,y)} \ \text{cantileverLSE}
\]

\[
x \ y \ w \ s_L \ e_L \ p \ n \ b_H \ b_e \ \theta_{(x,y)} \ \text{cantileverNLSE}
\]

**Figure 2.317:** Cantilever arrays of varying length. (a) Linear (cantileverLSE) and (b) nonlinear (cantileverNLSE) length variation from $s_L$ to $e_L$. 
The following constructor creates cantilevers with custom parameters. Structural origin is defined by the lower left corner at \((x,y)\). Cantilever length and widths are defined by the \(w_i\) and \(l_i\) values. Space between the adjacent cantilevers is defined by \(s_1, s_2, \ldots, s_n\). The base rectangle extends beyond the \(n^{th}\) cantilever’s right edge by the amount \(s_{\text{end}}\).

\[
x \ y \ s_1 \ w_1 \ l_1 \ s_2 \ w_2 \ l_2 \ \ldots \ s_n \ w_n \ l_n \ s_{\text{end}} \ b_H \ \theta_{(x,y)} \ \text{cantileverCustom}
\]

**Figure 2.318**: Cantilever array constructed with custom, user defined beam dimensions.
Below constructors create rectangular (with and without a triangular tip), trapezoidal and paddle cantilever structures. The rectangular base support anchor is defined by the anchor support width $a$ and GDS layer $L_a$. Structural layer is defined by the active layer $\theta_{(x,y)}$.

\[
\begin{align*}
&x \ y \ w \ \ l \quad b_H \ b_e \ a \ L_a \ \theta_{(x,y)} & \text{cantileverSR} \\
&x \ y \ w \ \ l \ t_H \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} & \text{cantileverSTri} \\
&x \ y \ w_a \ w_b \ l \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} & \text{cantileverSTrap} \\
&x \ y \ w_a \ w_b \ l_a \ l_b \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} & \text{cantileverSPaddle}
\end{align*}
\]

Figure 2.319: (a) Rectangular, (b) rectangular with a triangular tip, (c) trapezoidal and (d) paddle cantilever structures stemming from a rectangularly anchored base.
Curved and straight-circular beams where the curved segment is defined by the number of vertices $N_{\text{side}}$ and radius $R$.

\[ (x, y, w, R, N_{\text{side}}, b_H, b_{b}, a, L_{a}, \theta_{(x,y)}) \]

\[ (x, y, w, R, l, b_H, b_{b}, a, L_{a}, \theta_{(x,y)}) \]

Figure 2.320: (a) Curved and (b) straight-circular beams stemming from a rectangularly anchored base.
Below constructors create rectangular (with and without a triangular tip), trapezoidal and paddle cantilever structures. Constraints on parameter $h$ are $h < w/2$ and $h < w_d/2$.

$$x \ y \ w \ l \ h \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \ \text{cantileverHR}$$

$$x \ y \ w \ l \ t_H \ h \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \ \text{cantileverHTri}$$

$$x \ y \ w_a \ w_b \ l \ h \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \ \text{cantileverHTrap}$$

$$x \ y \ w_a \ w_b \ l_a \ l_b \ h \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \ \text{cantileverHPaddle}$$

**Figure 2.321:** Hollow (a) rectangular, (b) rectangular with a triangular tip, (c) trapezoidal and (d) paddle cantilever structures stemming from a rectangularly anchored base.
The following two constructors create hollow curved and straight-circular beams. In both cases $h < w/2$

$$x \ y \ w \ R \ N_{\text{sides}} \ h \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)}$$

**cantileverHCH**

$$x \ y \ w \ R \ N_{\text{sides}} \ l \ h \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)}$$

**cantileverHCF**

---

**Figure 2.322:** Hollow (a) curved and (b) straight-circular beams stemming from a rectangularly anchored base.
Below constructors create connected parallel beams. The gap ($g$) between the electrodes has the following constraint:

\[ g \leq \frac{l_B}{2} - 2w \quad \text{2 parallel beams} \]
\[ g \leq l_B - 2w \quad \text{3 parallel beams} \]

\[ x \ y \ w \ l_1 \ l_2 \ l_B \ g \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \ \text{cantileverPB2} \]
\[ x \ y \ w \ l_1 \ l_2 \ l_3 \ l_B \ g \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \ \text{cantileverPB3} \]

Figure 2.323: (a) Two and (b) three parallel, interacting beams.
U-Shaped spring designs with sharp rectangular, circularly filleted and circular links. Parameters \(a\) and \(L_a\) are anchor overlap and anchor GDS layer respectively.

\[
\begin{align*}
\text{cantileverUR:} & \quad x \ y \ w \ l_1 \ l_2 \ l_B \quad b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \\
\text{cantileverUCF:} & \quad x \ y \ w \ l_1 \ l_2 \ r \ N_{\text{sides}} \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \\
\text{cantileverUC:} & \quad x \ y \ w \ l_1 \ l_2 \ D \ N_{\text{sides}} \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)}
\end{align*}
\]

Figure 2.324: U-shaped (a) rectangular, (b) circularly filleted corners and (c) circular cantilever structures.
U-shaped structures with a central cantilever and a central paddle. Parameters $a$ and $L_a$ are anchor overlap and anchor GDS layer, respectively. Central cantilever length $l_2$ can be either positive or negative, respectively yielding a cantilever of width $w_2$ below or above the top link.

Figure 2.325: U-shaped structure with a center (a) cantilever and (b) paddle.
Below constructors create curved cantilever structures. The rectangular base support anchor is defined by the anchor support width \( a \) and GDS layer \( L_a \). Structural layer is defined by the active layer. Curved segments of the upper and lower paddle oscillators are defined by ellipses with radii \((r_x, r_y)\) and \((r_x', r_y')\), respectively.

\[
\begin{align*}
\text{cantileverCE} & : x \ y \ w \ l \ r_x \ r_y \ N_{\text{sides}} \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)} \\
\text{cantileverCEPaddle} & : x \ y \ w \ l \ r_x \ r_y' \ r_x' \ r_y' \ N_{\text{sides}} \ \omega_p \ l_p \ b_H \ b_e \ a \ L_a \ \theta_{(x,y)}
\end{align*}
\]

**Figure 2.326:** (a) Curved cantilever and (b) curved cantilever with a paddle.
2.10.6 Doubly Clamped Beams

Similar to the cantilever structures in figures 2.316a and 2.316b, these are doubly clamped beams with a linear and percentage variation with respect to the starting length \( s_L \).

\[
x y w s_L p n b_H b_c \Delta L \theta_{(x,y)} \text{ dcBeamL}
\]

\[
x y w s_L p n b_H b_c \text{ Percent } \theta_{(x,y)} \text{ dcBeamP}
\]

**Figure 2.327:** Doubly clamped beam arrays of varying length. (a) Linear (dcBeamL) variation from \( s_L \) with \( \Delta L \) increments and (b) Percentage (dcBeamP) length variation starting from \( s_L \).
Similar to the cantilever arrays in figure 2.317, these are doubly clamped beams with length variation bound between the starting ($s_L$) and the end length ($e_L$) values.

$$x \ y \ w \ s_L \ e_L \ p \ n \ b_H \ b_c \ \theta_{(x,y)} \ dcBeamLSE$$

$$x \ y \ w \ s_L \ e_L \ p \ n \ b_H \ b_c \ \theta_{(x,y)} \ dcBeamNLSE$$

**Figure 2.328:** Doubly clamped beam arrays of length varying between $s_L$ and $s_E$. (a) Linear ($dcBeamLSE$) and (b) nonlinear ($dcBeamNLSE$) length variation from $s_L$ to $e_L$. 
The following constructor creates doubly clamped beams with used defined parameters. The structure is similar to custom cantilevers in Figure 2.318.

\[
x \ y \ s_1 \ w_1 \ l_1 \ s_2 \ w_2 \ l_2 \ ... \ s_n \ w_n \ l_n \ s_{\text{end}} \ b_H \ \theta_{(x,y)} \ \text{dcBeamCustom}
\]

**Figure 2.329:** Doubly clamped array of beams constructed with custom, user defined dimensions.
Below constructors create rectangular and torsional doubly clamped beam structures. The rectangular base support anchor is defined by the anchor support width $a$ and GDS layer $L_a$. Structural layer is defined by the active layer.

\begin{align*}
  x & \quad y & \quad w & \quad l & \quad b_H & \quad b_e & a & \quad L_a & \quad \theta_{(x,y)} & \quad \text{dcBeamR} \\
  x & \quad y & \quad w_1 & \quad l_1 & \quad w_2 & \quad l_2 & \quad b_H & \quad b_e & a & \quad L_a & \quad \theta_{(x,y)} & \quad \text{dcBeamT}
\end{align*}

Figure 2.330: (a) Rectangular and (b) torsional doubly clamped beams.
Doubly clamped torsional beam.

\[ x \ y \ w_1 \ w_2 \ w_3 \ w_4 \ l_1 \ l_2 \ l_3 \ g \ b_H \ b_W \ a \ L_a \ \theta_{(x,y)} \ \text{dcBeamT2} \]

**Figure 2.331:** Doubly clamped torsional beam.
Doubly clamped array of $N$ interacting beams with lengths ranging linearly from $L_s$ to $L_e$.

Figure 2.332: Doubly clamped coupled beams.
Below constructor creates a curved doubly clamped beam structures. The rectangular base support anchor is defined by the anchor support width \( a \) and GDS layer \( L_a \). Structural layer is defined by the active layer \( L_d \). Curved segments of the upper and lower beams are defined by ellipses with radii \((r_{x1}, r_{y1})\) and \((r_{x2}, r_{y2})\), respectively.

\[
x \ y \ w \ l \ r_{x1} \ r_{y1} \ r_{x2} \ r_{y2} \ N_{sides} \ b_H \ b_e \ a \ L_a \ \theta(x,y) \quad \text{dcBeamC}
\]

Figure 2.333: curved doubly clamped beams.
2.10.7 Interacting Arrays

The following objects each have two constructors. The bracketed constructor (**<id** parameters ... **constructor>**) creates hierarchical structures that are instantiated within the current structure. The **id** parameter is a string prefix all generated structures for the particular object. This parameter is a unique continuous string of characters (no spaces or tabs). To ensure compatibility with the GDSII standard, the allowed structure character names are A−Z, a−z, 0−9, underscore (_), question mark (?) and dollar sign ($). Furthermore, the **id** parameter should be limited to no more than 15 characters, otherwise under certain circumstances, it’s possible to overwriting existing structures. Two major benefits of this constructor are that processing is faster and smaller files are generated. Both are a consequence of GDS structural hierarchy.

The second constructor is nearly identical with the exception of not having brackets, the string **id** parameter and the constructor suffix s. Furthermore, the constructor does not create any additional structures, hence resulting structures entirely are generated within the initialized structure. Consequently, the resulting files are bigger and take longer to generate.

In both cases, structures are generated with the following 3 GDS layers:

- **L<sub>YF</sub>** - front side beam layer
- **L<sub>YB</sub>** - back side release layer
- **L<sub>YM</sub>** - front side circular metal dot layer

```
<id x y N L<sub>1</sub> W<sub>1</sub> L<sub>2</sub> W<sub>2</sub> s H<sub>o</sub> L<sub>c</sub> H<sub>e</sub> L<sub>B</sub> H<sub>B</sub> d N<sub>s</sub> L<sub>yF</sub> L<sub>yB</sub> L<sub>yM</sub> θ<sub>(x,y)</sub> MARAs>
```

```
x y N L<sub>1</sub> W<sub>1</sub> L<sub>2</sub> W<sub>2</sub> s H<sub>o</sub> L<sub>c</sub> H<sub>e</sub> L<sub>B</sub> H<sub>B</sub> d N<sub>s</sub> L<sub>yF</sub> L<sub>yB</sub> L<sub>yM</sub> θ<sub>(x,y)</sub> MARA
```

![Diagram](image)

**Figure 2.334:** Two rectangular arrays of N interacting beams created using MARAs and MARA constructors.
Two trapezoidal array of $N$ interacting beams with identical parameters,

$$<id \ x \ y \ N \ L_W \ W_b \ s \ H_o \ L_c \ L_B \ H_B \ d \ N_s \ Ly_F \ Ly_B \ Ly_M \ \theta(x,y) \ MATALWs>$$

and with varying length ($L$) and widths ($W_a$ and $W_b$) between the two arrays.

$$<id \ x \ y \ N \ L_1 \ W_{1a} \ W_{1b} \ L_2 \ W_{2a} \ W_{2b} \ s \ H_o \ L_c \ L_B \ H_B \ d \ N_s \ Ly_F \ Ly_B \ Ly_M \ \theta(x,y) \ MATAs>$$

**Figure 2.335:** Two interacting trapezoidal beam arrays with (a) identical ($MATALWs$, $MATALW$) and (b) varying ($MATAs$, $MATA$) length and width parameters.
Array of two interacting rectangular beams.

\[
\langle \text{id } x \ y \ N \ L_1 \ W_1 \ L_2 \ W_2 \ s \ e \ H_o \ H_e \ L_s \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \rangle \text{ MAR2s} >
\]

\[
x \ y \ N \ L_1 \ W_1 \ L_2 \ W_2 \ s \ e \ H_o \ H_e \ L_s \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \rangle \text{ MAR2}
\]

Figure 2.336: Array of two interacting rectangular beams created using the MAR2s and MAR2, constructors.
Array of three interacting rectangular beams.

\[
\langle \text{id } x \ y \ N \ L_1 \ W_1 \ L_2 \ W_2 \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L y_F \ L y_B \ L y_M \ \theta_{(x,y)} \rangle \text{ MAR3s}
\]

\[
x \ y \ N \ L_1 \ W_1 \ L_2 \ W_2 \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L y_F \ L y_B \ L y_M \ \theta_{(x,y)} \text{ MAR3}
\]

Figure 2.337: Array of three interacting rectangular beams created using the MAR3s and MAR3 constructors.
Rectangular top array of elements interacting with single beams. The lower single beam elements are individually addressable via electrodes of width $H_w = W_1 + 2W_2 + 4s$ and height $H_e$.

$$<\text{id } x \ y \ N \ L_1 \ W_1 \ L_2 \ W_2 \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta(x,y) \ \text{MARC}s>$$

$$x \ y \ N \ L_1 \ W_1 \ L_2 \ W_2 \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta(x,y) \ \text{MARC}$$

Figure 2.338: Array of $2N$ rectangular beams interacting with $N$ single beams created using the MARCs and MARCnstructors.
Array of two interacting trapezoidal beams.

\[
\langle \text{id } x \ y \ N \ L_1 \ W_{1a} \ L_2 \ W_{2a} \ s \ e \ H_o \ L_s \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \ \text{MAT2s} \rangle
\]

\[
x \ y \ N \ L_1 \ W_{1a} \ L_2 \ W_{2a} \ s \ e \ H_o \ L_s \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \ \text{MAT2}
\]

**Figure 2.339:** Array of two interacting trapezoidal beams created using the **MAT2s** and **MAT2** constructors.
Array of three interacting trapezoidal beams.

\[
\begin{align*}
\text{id} & \ x \ y \ N \ L_1 \ W_{1a} \ W_{1b} \ L_2 \ W_{2a} \ W_{2b} \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \ \text{MAT3s}
\end{align*}
\]

\[
x \ y \ N \ L_1 \ W_{1a} \ W_{1b} \ L_2 \ W_{2a} \ W_{2b} \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \ \text{MAT3}
\]

Figure 2.340: Array of three interacting trapezoidal beams created using the \text{MAT3s} and \text{MAT3s} constructors.
Trapezoidal top array of elements interacting with single beams. The lower single beam elements are individually addressable via electrodes of width $H_w = W_{1a} + 2W_{2b} + 4s$ and height $H_e$.

\[ <id x \ y \ N \ L_1 \ W_{1a} \ W_{1b} \ L_2 \ W_{2a} \ W_{2b} \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \ MATCs> \]

\[ x \ y \ N \ L_1 \ W_{1a} \ W_{1b} \ L_2 \ W_{2a} \ W_{2b} \ s \ e \ H_o \ H_e \ L_B \ H_B \ d \ N_s \ L_yF \ L_yB \ L_yM \ \theta_{(x,y)} \ MATC \]

**Figure 2.341:** Array of $2N$ trapezoidal beams interacting with $N$ single beams created using the *MATCs* and *MATC* constructors.
Two interacting rectangular arrays between elements of differing lengths. The electrode base varies linearly from the value of $H_e$ to $H_e + \Delta$.

\[ <id \ x \ y \ N \ L \ \Delta \ W \ s \ H_o \ L_o \ H_e \ L_B \ H_B \ d \ N_s \ L_1 \ L_2 \ L_3 \ \theta_{(x,y)} \ MARALINEARs> \]

\[ x \ y \ N \ L \ \Delta \ W \ s \ H_o \ L_o \ H_e \ L_B \ H_B \ d \ N_s \ L_1 \ L_2 \ L_3 \ \theta_{(x,y)} \ MARALINEAR \]

Figure 2.342: Two interacting rectangular arrays between elements of differing lengths. Beam lengths vary linearly across the electrode base. Structures are generated using the MARALINEARs and MARALINEAR constructors.
Two rectangular arrays between elements of differing lengths. The electrode base has a slow non-linear variation from $H_e$ to $H_e + \Delta$. The non-linear element was constructed using a Bezier curve.

\[
\begin{align*}
\text{id} & \quad x \quad y \quad N \quad L \quad \Delta \quad W \\
& \quad s \quad H_o \quad L_o \quad H_e \quad L_B \quad H_B \quad d \quad N_s \quad L_1 \quad L_2 \quad L_3 \quad \theta_{(x,y)} \quad \text{MARACURVES}\n\end{align*}
\]

\[
\begin{align*}
x & \quad y \quad N \quad L \quad \Delta \quad W \\
& \quad s \quad H_o \quad L_o \quad H_e \quad L_B \quad H_B \quad d \quad N_s \quad L_1 \quad L_2 \quad L_3 \quad \theta_{(x,y)} \quad \text{MARACURVE}
\end{align*}
\]

Figure 2.343: Two interacting rectangular arrays between elements of differing lengths. Beam lengths vary non-linearly across the electrode base. Structures are generated using the MARACURVES and MARACURVE constructors.
2.10.8 Stress, Strain Measurement Structures

2.10.8.1 Guckel Rings

Guckel ring structures are used to estimate residual stress in structural layers.

\[ x \ y \ r_W \ R \ N_{sides} \ b_W \ b \ L_c \ W_c \ a \ L_a \ \theta_{(x,y)} \]

Figure 2.344: Schematic of a Guckel ring structure used for measuring residual stress.

Below is a Guckel ring array structure. The varying radii range from \( R_s \) to \( R_e \) in increments of \( \Delta R \).

\[ x \ y \ r_W \ R_s \ R_e \ \Delta R \ N \ b_W \ b \ L_c \ W_c \ a \ L_a \ \theta_{(x,y)} \]

Figure 2.345: Schematic of a Guckel ring array structure with varying radius.
2.10.8.2 Diamond Ring

\[
(x, y, w_1, w_2, w_3, l_1, l_2, l_3, b_H, b_W, a, L_a, \theta_{(x,y)}) \text{ diamondRing}
\]

Figure 2.346: Stress measurement diamond ring.
2.10.9 Suspended Fluid Cell

Fluid cell ($L_c$) with a suspended region defined by radius $r_3$ and a spiral delay line heater ($L_b$). Central red dot indicated coordinate ($x,y$). The fluid input funnel shape are defined by the shapeReso parameter.

$\begin{align*}
  x & \ y & \ w_1 & \ w_2 & \ w_2T & \ w_3 & \ w_4 & \ L_1 & \ L_2 & \ r_1 & \ r_2 & \ r_3 & \ N_{\text{sides}} & \ a & \ b & \ c & \ d & \ e & \ f & \ g & \ h & \ i & \ L_a & \ L_b & \ L_c & \ L_d & \ L_f & \ \theta_{(x,y)} & \text{fluidCell}
\end{align*}$

Figure 2.347: Fluid cell ($L_c$) with an integrated spiral delay line heater ($L_b$) and release trenches ($L_e$).
3.1 Basic Shapes

3.1.1 Pillar-Hole (Square/Hex) Array

Within this module, polygons are created and arrayed. User defined parameters consist of number of sides, size and rotation of individual elements, the GDS layer number, and lower left position, number of elements, and pitch between the arrayed structures. This module creates cells for individual elements used to create pillars and holes. Pillars are created by performing a boolean operation between a box of dimensions \((\text{Pitch}_x \times \text{Pitch}_y)\) and the specified shape. The shapes are then instantiated within 4 arrayed cells consisting of square and hexagonal arrays of both polarities. Figures 3.348\((a - h)\) show several examples of the pillar hole array.

3.1.2 Torus

This module creates arc shapes defined by the start \((\theta_s)\) and end \((\theta_e)\) sweep angles of the individual elements. Torus is created by sweeping an arc over 2\(\pi\) radians, i.e. \(\theta_e - \theta_s = 360\) degrees. Default parameters for the torus module are \(\theta_s = 0\) degrees and \(\theta_e = 360\) degrees. Generated features are defined by the number of sides for the inner and outer arc segments, the GDS layer number, and rotation of individual shapes. Position, elements and pitch parameters define square and hexagonal array elements. Figures 3.348\((i - j)\) show 4 examples of various arcs.

3.1.3 Grating Coupler - Bulls Eye

The module creates concentric arcs to form grating couplers or bulls-eye patterns. The arc section is defined by a sweep angle and arc width. The arrayed elements are located at the specified \((x, y)\) position. First element is placed at a distance defined by the radius midpoint. Parameter Rings defines the number of arrayed elements instantiated at the specified pitch. Figures 3.348\((k - l)\) show several examples of grating coupler structures.
3.1.4 Spirals

Spirals are defined by their center position, GDS layer number, shape width, and number of turns. Shape resolution is specified by the Increment parameter. Choosing many turns with a small increment (high resolution) will increase the GDS file generation time. Spiral structures are generated and placed into a GDS structure named top.

We offer three types of spirals, Archimedes, Fermat and Logarithmic (Figures 3.348(m−o)). Archimedes spiral has a uniform spacing between turns (s) and is defined as

\[ r = m\theta \]  

(3.23)

where

\[ m = \frac{s + \text{width}}{2\pi} \]  

(3.24)

Fermat spiral is defined as

\[ r = \sqrt{a^2 \theta} \]  

(3.25)

and Logarithmic spiral is defined as

\[ r = ae^{b\theta} \]  

(3.26)

3.1.5 Gratings

Gratings module generates user-defined lines arrayed at a predefined pitch. Each line of the table will create two cells, one for the line and the other for the grating array. Generated structure names will have a prefix RX where X represents the table row number. The load table button allows users to upload text files. Values from loaded text files will populate the table. Top cell will have an instance of each array separated by the TopCellSpacing value. Figures 3.348(p) shows an example of a grating structure. An example of a grating text file is located \loadFiles\gratingTable.
Figure 3.348: Various examples from the BasicShapes branch. Rectangular arrays of (a,e,g) holes and (b,f,h) pillars. Hexagonal array of (c) holes, (d) pillars, (i) torii and (j) arcs. (k) Grating coupler and (l) bulls-eye pattern. Spirals created using various parameters (m) Archimedes, (n) Fermat, (o) and Logarithmic. (p) Grating structure.
3.2 Lithography Machine Resources

3.2.1 CNST Reticle Frame Generator

This module creates reticle frames for the CNST Nanofab ASML-PAS5500 i-line stepper and contact aligners. Default stepper option will print reticle marks, user defined label and barcode, NIST and CNST logos. ASML barcode and label are limited to 12 and 14 characters respectively. Two remaining options are for 5 inch and 7 inch contact masks. In both cases, label, along with NIST and CNST logos are printed near the periphery of the mask. Contact labels are limited to 22 characters.

3.2.2 Generic Reticle Frames

Within this module, reticle marks, label and barcode are generated for several steppers.

3.2.3 Ebeam Lithography - Job and Schedule File Generator

This module generates job and schedule files for the CNST JEOL 6300 electron beam lithography tool. The module offers a number of graphical features that help ensure proper pattern and alignment mark placement. Additionally, it features a wide variety of dose variation features including base dose matrices, single and multi-shot rank modulation, user defined dose table modulation, and several dose ramping functions. This node has 4 sections accessible using the tabs located at the top of the panel named Main, Align, Pattern and DoseMatrix.

3.2.3.1 Default Values Initialization File

Many features within the CNST JEOL 6300 JDF-SDF module are customizable using the provided CNST_defaultValues.xml initialization file. For example, position size, auto-loader (ALD) position, paths, calibration parameters, alignment marks represent a few parameters that are stored within the initialization file. If this file is missing, hardcoded values will be displayed within the toolbox. Also, directly to the left of the About button within the Main tab, a string will indicate if values were loaded from the initialization file.

3.2.3.2 Main

Within the Main panel material parameters, jdf and sdf file names, alignment schemes, paths and calibration parameters are preset. The PATH and
CALPRM variables are editable. Most of these parameters are predefined within the CNSTdefaultValues.xml initialization file. Figure 3.349 shows parameters under the Main panel.

![Main panel showing various available options under Main tab.](image)

**Figure 3.349: Main panel showing various available options under Main tab.**

### 3.2.3.3 Align

The alignment panel allows users to define position and size of global \((P\) and \(Q\)) and local \((M_1, M_2, M_3, \text{ and } M_4)\) alignment marks. The positions are in wafer coordinates in units of micrometers. Under mark definition, mark width and length values are specified in micrometers for the the global \(P\) (GLMP), global \(Q\) (GLMQRS) and chip (CHMARK) alignment marks. The Align section is only used if global and local marks are selected under the Alignment field within the Main tab. The left-most alignment option activates the global marks, while the right-most option controls local (chip) alignment. The options are by default inactive as indicated by the OFF state. Figure 3.350 shows parameters under the Align JEOL 6300 panel.

### 3.2.3.4 Pattern

This section defines pattern arrays within the job definition file. Pattern files specified, arrays are generated from those pattern files and stored into a vector array is accessed when job and schedule definition files are created. To add a pattern file to the database, type in a pattern file name then click the AddFile button. A string label will appear and indicate that the entered .v30 file was added to the pattern file list. More files are added by repeating the previous process. Entered pattern names will appear in the pull down selection box.
Select a pattern, then type underneath the ARRAY section, type in the starting coordinates of the first element of the array, then number of elements and the pitch between elements, then type in the dose modulation for the pattern file. For a single element leave both Element values at 1. Click the Generate Array button. The value of the generated array will appear in a selection box directly to the left of the Generates Arrays field. Repeat the process, i.e. choose file name, then pattern position, number of array elements, pitch between elements, enter a dose modulation and click the GenerateArray button. Delete button to the right of the selection box within Generates Arrays field allows users to delete previously generated arrays stored within the selection box. Also, generated arrays can be modified by choosing an array from the saved list. Following parameter modifications, clicking the UpdateArray button overwrites the previous entry. Figure 3.351 shows parameters under the Pattern panel.

To view the arrays as well as the job and schedule files at any point during the array creation, click the Show SDF JDF button. A new frame will pop-up with a text area containing the contents of the jdf and sdf files. Another frame will pop-up and will show graphically where patterns are placed (Figure 3.352). The feature is similar to the JEOL Array Check program. Following any alteration of the generated array elements, clicking the Show SDF JDF button will update the job and schedule file panel as well as the array check panel.

### 3.2.3.5 Dose Matrix

Dose matrix module is a means to create complex dose and base dose matrices. The procedure initiates by choosing a pattern file name, the position, number of elements and pitch between the elements of the resulting dose array (Figure 3.353). There are various options under the DoseMethod tab. Standard DoseMatrix is the default method for executing a dose matrix on the chosen V30 file. Method BaseDose is used to determine a proper base dose for
Figure 3.351: Pattern panel showing user-defined pattern files, array definitions, and generated arrays.

Figure 3.352: Display showing user generated pattern arrays and global mark positions (P and Q).
a proximity effect corrected pattern. Here base dose (RESIST parameter) is modified within the schedule file, the pitch in the two directions will represent the OFFSET parameter, with the RESIST value derived from the Start text field within this tabbed panel. There are 4 options within the Ramp selection menu. Linear and Percent options use respective right-most text-field values of Linear and Percentage to increment the dose array from the starting value represented by the Start text field. For instance, choosing Linear with a value of 10 within the Linear text field in conjunction with a Start value of 100 for an array of 4 elements, percentage modulation dose values will be 100, 110, 120 and 130. Analogously, with an active Percent ramp option and a value of 10 within the Percentage text field in conjunction with a Start value of 100 for an array of 8 elements, percentage modulation dose values will be 100, 110, 121, 146.41, 161.05, 177.16 and 194.87. LinearSE and NonlinearSE are two ramp methods that utilize number of arrayed elements, the Start and End text field values to determine the percentage dose modulation.

Figure 3.353: Dose matrix panel showing various user defined pattern, array and dose parameters.

Dose represents the default value of the Type selection. DoseTable option under Type uses values from the DoseTable tabbed panel. Within the Dose Table tabbed panel, values for the shot rank and modulation are either manually entered or uploaded from a text file (Figure 3.354). The method assumes the standard dose modulation table format, as seen in the layout beamer proximity effect corrected output file with extension JDI. An example dose matrix JDI file is located in the \loadFiles\jeol6300DoseTable directory. When using the table method, ramp can either be the default Linear or Percent option. Depending on which method is chosen, the respective Linear and Percentage text fields are used to modify the individual modulation values from the table. The sdf and jdf
files could be shown or generated using the respective buttons at the bottom of the panel.

Figure 3.354: Dose matrix panel showing various user defined pattern, array and dose parameters.

3.2.4 EBL Alignment Offset

The JEOL alignment module calculates the offsets between the designed and observed wafer positions. Type in the designed mark positions of the P (or Q) marks. This value is specified in material (or wafer) coordinates with micrometer units. Check the substrate holder center position and type in those values for the wafer center. Once the marks are located in the scanning electron microscope (SEM) of either of the CNST ebeam tools, type in the stage coordinates of this observed mark position, then press Enter or click the Calculate button. Within the OFFSET text field, calculated values will appear. These values should be used within the SETWFR routine to manually check global alignment. Furthermore, these values are placed within the schedule file (.sdf file) as:

\[
OFFSET(X_{offset}, Y_{offset})
\]

3.2.5 EBL Max Clock

This module calculates the maximum writing frequency and the minimum shot time for the 4\(^{th}\) and 5\(^{th}\) lens electron beam lithography tools. Calculation is based on beam current, writing dose, shot pitch and tool type.

3.2.6 EBL Write Time Estimation

This module calculates an estimated electron beam lithography write time. Calculation is based on beam current, die area (written die area in micrometers squared - a value easily obtained from various CAD and fracturing tools), writing
dose, fields per die and the number of die. Stage time is ‘estimated’ at 1 second per stage move. This is not quite correct (‘estimate’) since stage motion depends upon distance traveled and the EBL tool. The estimated calculation does not take into account tool calibration time.
3.3 Advanced CAD Resources

Many of the modules in the advanced CAD resources are implemented within CNST scripting. Therefore, many modules refer to sections and figures within the scripting command reference chapter. Users should note that scripting offers more flexibility with many more features not implemented within its GUI counterpart.

3.3.1 Label Maker

Label Maker module is used to generate chip labels into a GDS file that could then be instanced into a user defined pattern. There are 4 label type options. The automated Outer and Row−Column will generate number arrays, whereas the custom variants are more versatile allowing imported tab delimited data to be cast as labels. Two test label files are included in \loadFiles\labels example directory. The files were generated in Excel and exported as plain text tab delimited.

Within the label maker module, first, number of elements, pitch between the elements, font specifications and label type are specified. Font resolution is the rendering resolution of the Bezier curves that construct font shapes. This parameter is defined as shapeReso within CNST scripting (see Section 2.2.6 and Figure 2.16). The four label types are described within the label maker scripting command reference guide. This module does not implement automatic letter labeling. Scripting section 2.6.3 further illustrates automatically generated Outer and Row−Column (Figure 2.45a,b) and the respective custom, user-defined labels (Figure 2.45c,d).

3.3.2 Text To GDS

Text to GDS module renders strings composed of vector fonts to GDS shapes. Font resolution parameter controls the rendered shape resolution. Example of font resolution and generated text are shown respectively in Figure 2.2.6 and 2.43.

3.3.3 Arbitrary Function Generator

Arbitrary Function Generator module will create a GDS file with a shape defined by a mathematical equation. User will input a lower and upper bound, an increment and line width for the function. Within this module, users define a function \( f(x) \) using the lower case variable \( x \). Functional definitions follow the Java Math class. Scripting section 2.7.4 offers a more efficient function generator. This section also describes how functions are represented.

Functions constructed with a large number of points could take a con-
able amount of time to cast the resulting GDS file. Label indicator above the module buttons will display the path and file name once it’s finished writing data to file. Also, casting a curved function defined by a large width could result in a shape with looped interiors.

### 3.3.4 Binary Zone Plate

In a binary zone plate, zones switch from opaque to transparent at radii,

\[
r_n = \sqrt{n \lambda f + \frac{n^2 \lambda^2}{4}}
\]

(3.27)

where \( n \) is an integer, \( \lambda \) is the wavelength and \( f \) is the focal distance measured from the center of the zone plate. Each shape is constructed with double the number of prescribed sides (interior and exterior) whereas the central element (circle) vertices are equal to the \( \text{Sides} \) module parameter. Figure 3.355 shows two examples of binary zone plates. Zones are either composed of a single user defined GDS layer, or cast into zone-respective GDS layers. This implies that zone at a radius \( r_n \) is rendered to GDS layer number \( n \) where \( n = 0, 1, 2, \ldots 255 \).

Due to 8-bit layer GDS standard restrictions, zone numbers above the upper bound (255) reset the counter back to GDS layer number 0. Consequently, zones at radii \( r_{256}, r_{257}, r_{258}, \ldots \) would respectively have GDS layer numbers 0, 1, 2, ...

![Figure 3.355](image)

**Figure 3.355:** GDS output of a zone plate with 10 zones, 88 sides per zone, \( \lambda = 632.0 \text{nm} \), and \( f = 0.001 \text{m} \). (a) Single layer zone plate. (b) Enabling the different layers option casts each zone to a respective GDS layer number.
3.3.5 Photonic Crystals

Photonic crystals module creates two hexagonal arrays of circles. Circles are defined by a diameter \(d\), number of sides and GDS layer number parameters. Within the array circular elements are defined by the spacing proximity to the nearest neighbor \(h\). Hexagonal arrays are defined by the separation between the two arrays \(s\), and the array size \(S_x\) and \(S_y\) along the corresponding \(x\) and \(y\) directions (Figure 3.356). Table of values are either manually populated or data is loaded from a text file. Example photonic crystal table file is included in the \loadFiles\photonicCrystalTable directory. Each table value is tab separated, each table row starts on a new line. Figure 3.356 shows various parameters from the photonic crystal module.

![Photonic crystal example of a rendered GDS structure labeled with module parameters.](image)

3.3.6 Random Polygons

Random Polygons module will randomly place polygons into a user defined area. Separation parameter defines the minimum separation between the outer radial perimeter of the objects. Polygons are defined by the outer radius and number of sides. The polygon placement area is defined by the \textit{Width} and \textit{Length} parameters. If the randomly generated coordinate violates the minimum separation distance, the module will keep generating random coordinates until the maximum number of failures (defined by the parameter \textit{Iteration}) is reached.

By clicking the \textit{RandomRotation} checkbox, each shape will be randomly rotated within the top cell. Enabling the option for features with many sides is
not useful, however, with \( \text{Sides} = 3 \) and random rotation produces randomly oriented triangles within the array (see Figure 2.78 in section 2.7.11).

This shape could be generated using the scripting constructor \texttt{randomPolygons}, as described in section 2.7.11.

### 3.3.7 Random Rectangular Array

Random rectangular array module will randomly place squares or rectangles into a user-defined area specified by the number of elements and pitch (in \( x \) and \( y \) directions). The shape size (square or rectangular) as well as the probability that a lattice point is occupied are user-defined parameters. Furthermore, user can load \( x \) and \( y \) coordinate pairs and cast them to lattice sites. This module was suggested by a researcher and used for generating disordered metamaterial structures.

This module allows users to load \( x \) and \( y \) coordinates where objects will be drawn. The ASCII file should be arranged with \( x \) and \( y \) pairs tab or space separated, with one pair per line. The first line of the data file is defined as a header and is consequently skipped. An example file is located under the `\loadFiles\randomRectangularArray` directory. To use the module, first choose the size of the square or rectangular object, then check the Load File check box, then click Load File, navigate to a directory with the text file, click on the file, then click OK. The chosen filename will be displayed in the label field directly below. Then click Create GDS to cast the shapes into a GDS file. NOTE: when loading from file, this module ignores the number of elements field, and will not display the total number of drawn shapes, skipped shapes, etc, within the results label field.

### 3.3.8 Cantilever Arrays

Cantilever Arrays module will create a GDS file with a modulated (linearly, non-linearly, percentage, sinusoidally, etc) cantilever array structures. Number of resulting structures is defined by the \textit{Elements} parameter. Levers are characterized by a width, pitch and variable length values. Cantilevers are connected to a rectangular base of a user specified height. The width of the base rectangle is defined by the number of elements, pitch and the base extent.

By choosing \texttt{Linear} or \texttt{Percentage} type variation, the generator will utilize the \texttt{Length start} value and will increment the length using the linear or percentage parameters respectively. Type \texttt{Linear SE} and \texttt{non-Linear SE} use the \texttt{Length start} and \texttt{Length end} values and respectively vary the cantilever lengths linearly and non-linearly. Type \texttt{Sinusoid} will use the \texttt{Length start} value as the initial length value and will vary the cantilever lengths sinusoidally over a \( 2\pi \) period with an amplitude defined by the \textit{sin Amplitude} value.

The MEMS-NEMS library offers identical elements (section 2.10.5) as well as many other shapes with enhanced functionality. Figure 2.316 illustrates can-
tilevers with linear, percentage, and sinusoidal length modulation. Figure 2.317 shows examples of a linear and exponential length variation using start and end length values.

### 3.3.9 Verniers

This module generates verniers between two specified alignment layers. User defined parameters consist of GDS layer numbers for layer A and B, vernier resolution, number of vernier tick lines, text labels for the two layers, vernier line width, length and pitch. Table of values are either manually populated or data is loaded from a text file. Example vernier table file is included in the \loadFiles\vernierTable directory. Each table value is tab separated, each table row starts on a new line. Vernier example is shown in the scripting command reference section 2.8.5 and figure 2.94.

### 3.3.10 Fractals

This module contains Sierpinski triangle and carpet, curved tree, and the Vicsek saltire and cross fractals. Each fractal iteration is stored into a GDS structure with a prefix cell name concatenated with the iteration number. The extent of the final structure is defined by the module length and width parameters. Fractal examples are defined in the scripting command reference section 2.7.3 (see also Figure 2.58).

### 3.3.11 Grayscale Image To GDS

Grayscale Image to GDS module will convert a 8-bit gray scale BMP (PNG or JPG) and cast it to a GDS file where each level of gray (values 0 to 255) will be mapped to corresponding GDS layer numbers (GDS layers 0 to 255). This mapping is important for grayscale electron beam lithography. In this case, the generated file can be used in conjunction with a resist contrast curve to print three dimensional structures [104].

Resulting data is stored into a GDS structure named top. Within the module, data is read and converted line by line. Adjacent pixels of same value are merged together in order to minimize the resulting GDS file size. Also note that data within the top cell is not centered around the origin. The lower left hand corner of the image is mapped to the origin of the GDS file (point (0,0)). The image is mapped within the first quadrant (+x, +y) of the GDS file.

If the BW option is checked, then all color values other than white (pixel value 255) will be cast to GDS layer 1. In this case, adjacent pixels are merged, however, if 2 pixels are of different value, they will be both cast to GDS layer 1, as 2 different GDS layer 1 pixels. To have pixels merged, use a graphic editing software package to cast image to black and white BMP, then change the mode to RGB 8-bit gray scale. Figure 3.357 shows an image and the resulting GDS
files.

![Figure 3.357: Grayscale to GDS. (a) 8-bit gray scale image (in this case black and white). The resulting GDS file (b) without and (c) with the BW option enabled.](image-url)
4.1 Programming Examples

The nanolithography toolbox distribution includes a NetBeans 8.0.2 project entitled CNSTprogrammingExamples. The src directory contains a package with source .java example files. These files could be used with any integrated development environment (IDE) in conjunction with Java 8. Furthermore the nanolithography toolbox jar file must be included within the project compile-time libraries. The following examples illustrate access to existing scripting methods through programming. Furthermore, the project directory includes the output GDS example files.

The following is a procedure for setting up the included Netbeans example project:
1. Start NetBeans -> File -> Open Project... -> navigate to \EXAMPLES\Programming and open the CNSTprogrammingExamples project
2. To fix the broken library link, right click on the project and choose Properties, click Libraries and remove the red-error highlighted compile-time library. Click the "Add JAR/Folder" button, then choose the CNST Nanolithography Toolbox JAR file and click OK.

4.1.1 Template

```java
package cnstprogrammingexamples;
import CNST.Scripting;
import JGDS2.*;
import java.io.File;

public class Example01Template {
    public static void main(String[] args) {
        Lib lib = new Lib();
        // Insert Code Here
        File f = lib.GDSOut("Example01Template.gds");
    }
}
```
The above example is a template for java files accessing the CNST nanolithography toolbox. Lines 3 and 4 import all the objects needed to access scripting methods and create the GDS file. Line 5 import statement allows file input-output access. The example class contains a main method. Line 14 initializes the GDS library. Line 18 sets the GDS output file. The print statement (line 19) is optional but helpful since it displays the full path of the saved GDS file.

### 4.1.2 Script Method Access

The programming reference provides method headers for the available scripting methods within the toolbox. These methods either return a GDS Area object (GArea) and vector array of these objects (ArrayList<GArea>), or are void in which case the the areas are placed within a particular GDS structure.

```java
package cnstprogrammingexamples;
import CNST.Scripting;
import JGDS2.*;
import java.io.File;
/
/**
 * @author rob
 */
public class Example02ScriptingMethodAccess {

    public static void main(String[] args) {
        Lib lib = new Lib();
        //
        // create a GDS struct SingleEllipse
        Struct ellipse = new Struct("ellipseSingle");
        // add a GArea of an ellipse from the nanolithography toolbox scripting method
        ellipse.add(Scripting.createEllipse(0, 0, 0.2, 0.5, 44, 50, 4));
        // create a GDS struct to store many instantiated ellipses
        Struct manyEllipses = new Struct("InstantiatedEllipses");
        // instantiate ellipses along a circular path
        double circleRadius = 20; // units in micrometers
        int numberOfInstances = 100;
        double increment = 2 * Math.PI / numberOfInstances;
        for (int i = 0; i < numberOfInstances; i++) {
            manyEllipses.add(new Ref(ellipse, circleRadius * Math.cos(i * increment), circleRadius * Math.sin(i * increment)));
        }
    }
}
```
The above example creates a GDS structure object ellipse (line 17) with a structure name ellipseSingle. Line 19 adds a GDS area of an ellipse into the structure object. Here, the ellipse is created by one of the available nanolithography toolbox scripting methods using the following method header (Reference Methods section shows all the available methods)

```
public static GArea createEllipse(double x, double y, double radiusX,
                                double radiusY, int numSides, double THETA, int gdsLayer)
```

Line 22 creates a GDS structure object manyEllipses. Using a for loop the ellipse object is instantiated along a circular path using the Ref constructor (line line 27). Figure 4.358 shows the output of the instantiated ellipse object. Lines 31 and 32 add the two structures to the GDS library. The following are Ref constructors

```
Ref(Struct structure, double x, double y)
Ref(Struct structure, double x, double y, int mirror)
Ref(Struct structure, double x, double y, int mirror, double angle)
Ref(Struct structure, double x, double y, int mirror, double mag,
    double angle)
```

To use mirroring within Ref the existing class must implement an interface Const, then set int mirror to MIRROR. Please consult the JGDS user manual for examples and more information on GDS structure instantiation [76].

![Programming example illustrating instantiation of objects created using the nanolithography scripting methods.](image)
4.1.3 Boolean Operations and Affine Transformations

The following example illustrates affine transformations of GDS areas and boolean operation between GDS areas. The code first generates a GDS structure (line 18) to store boolean subtraction between a circle and a circle wave object. Lines 19 and 20 show instantiation of GDS areas using the available nanolithography toolbox methods. Line 21 subtracts the circle wave from the circle object. Subsequently the result (circle) is stored in the booleanCCW GDS structure. Figure 4.359 shows the rendered GDS shapes.

![Figure 4.359](image)

**Figure 4.359:** Rendered GDS output shape resulting from a subtraction between a circle and a circle wave object. The black dots represent shape vertices.

Lines 25-30 create a GDS structure and two circular areas, then add the areas to the structure. A copy of circle1 into a temp area object occurs in line 34. Directly following, in line 35 a boolean OR operation with circle2 occurs. Line 37 shows a GDS area translation using built in Java affine transformations. In a similar fashion, the areas could be scaled, rotated and sheared. Lines 40 to 59 repeat the process for a boolean AND, XOR, and SUBTRACT operations. Figure 4.360 shows the output of the rendered structures.

![Figure 4.360](image)

**Figure 4.360:** Rendered GDS output shape of the (a) original two circles and various boolean operations between the areas (b) OR, (c) AND, (d) XOR and (e) SUBTRACT.

```java
package cnstprogrammingexamples;
```
import CNST.Scripting;
import JGDS2.*;
import java.awt.geom.AffineTransform;
import java.io.File;

/**
 * @author rob
 */
public class Example03BooleanOperationsTransformations {

    public static void main(String[] args) {
        Lib lib = new Lib();
        // // Subtraction of 2 GAreas - circle - circleWave
        Struct booleanCCW = new Struct("BooleanCircleCircleWave");
        GArea circle = Scripting.createEllipse(0, 0, 14, 14, 44, 0, 4);
        GArea circWave = Scripting.createCircleWave(0, 0, 10, 10, 0.5, 128, 0, 4);
        circle.subtract(circWave);
        booleanCCW.add(circle);

        // boolean examples between 2 circles
        Struct booleanCircles = new Struct("BooleanCircles");
        GArea circle1 = Scripting.createEllipse(0, 0, 1, 1, 44, 0, 4);
        GArea circle2 = Scripting.createCircleWave(1, 1, 1, 1, 44, 0, 4);
        // add both circles to the struct
        booleanCircles.add(circle1);
        booleanCircles.add(circle2);

        // OR Example
        GArea temp = new GArea(circle1);
        temp.or(circle2);
        // affine transform - translation 4um in x
        temp.transform(AffineTransform.getTranslateInstance(4, 0));
        booleanCircles.add(temp);

        // AND Example
        temp = new GArea(circle1);
        temp.and(circle2);
        // affine transform - translation 8um in x
        temp.transform(AffineTransform.getTranslateInstance(8, 0));
        booleanCircles.add(temp);

        // XOR Example
        temp = new GArea(circle1);
        temp.xor(circle2);
        // affine transform - translation 12um in x
        temp.transform(AffineTransform.getTranslateInstance(12, 0));
        booleanCircles.add(temp);

        // SUBTRACT Example
        temp = new GArea(circle1);
        temp.subtract(circle2);
        // affine transform - translation 16um in x
        temp.transform(AffineTransform.getTranslateInstance(16, 0));
4.1.4 Labeled Arrays

Labeled arrays could be created in a variety of ways using the nanolithography toolbox scripting methods. For instance, general areas could be instantiated using any of the available array methods. The following example uses a for loop to iterate through various diameters and separations with

```java
public static void createSqrHoleC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer)
```

to create an array of circular objects, centered at position \((x, y)\). Within the loop, labels for each array are created using the centered text method

```java
public static GArea createTextGdsC(String textString, String fontName, double fontSize, double x, double y, double THETA, int gdsLayer, double shapeReso)
```

The program starts by initializing a GDS structure `top` (line 16). This structure will be used to hold instantiated arrays of various circular apertures. Lines 18-29 initialize a variety of variables used to create the arrayed structures. The nested for loop (lines 31-40) iterates through diameters and separations between circular array elements. In the inner loop, a pitch is calculated in line 33. Directly following, a string defining the GDS structure name of individual arrays is established (line 34). Number of array elements, based on the array extent and pitch, are calculated (line 35). Then `createSqrHoleC` method is used to create and instantiate the hole array patterns within the GDS structure `top` (line 36). A string label is defined using the diameter and pitch values (line 37) and a GDS area containing the text label is added to the GDS structure `top` (line 38). Figure 4.361 shows the rendered GDS output.
Figure 4.361: Programming example of labeled arrays.

```java
public class Example04LabeledArrays {
    public static void main(String[] args) {
        Lib lib = new Lib();
        //
        Struct top = new Struct("top");
        // initialize variables
        double[] diameter = new double[]{1, 2, 3, 4};
        double[] separation = new double[]{0.25, 0.5, 1, 2};
        double arrayExtent = 750;
        double arrayPitch = 2000;
        String s, lbl;
        double pitch;
        int numSides = 44;
        int numElements;
        int gdsLayer = 4;
        double labelOffset = 200;
        double fontSize = 170;
        double shapeReso = 0.1;
        // for loop to create arrays and labels
        for (int i = 0; i < diameter.length; i++) {
            for (int j = 0; j < separation.length; j++) {
                pitch = (diameter[i] + separation[j]);
                s = "circle_d" + (int) diameter[i] + "um_p" + (int) (pitch * 1000) + "nm";
                numElements = (int) (arrayExtent / pitch);
                Scripting.createSqrHoleC(s, top, i + arrayPitch, j + arrayPitch, diameter[i] / 2, diameter[i] / 2, numSides, pitch, pitch, numElements, numElements, 0, gdsLayer);
                lbl = "d=" + (int) diameter[i] + "um; p=" + (int) (pitch * 1000) + "nm";
                top.add(Scripting.createTextGdsC(lbl, "Arial", fontSize, i + arrayPitch, j + arrayPitch - labelOffset - arrayExtent / 2, 0, gdsLayer, shapeReso));
            }
        }
        lib.add(new Ref(top, 0, 0));
        //
        File f = lib.GDSOut("Example04LabeledArrays.gds");
        System.out.println("Saved to " + f.getAbsolutePath());
    }
}
```
4.1.5 MEMS Perforated Flexures

The following code creates a perforated MEMS flexure. The scripting element is called `flexure2C` and described in subsection 2.10.4. The code describes storing vector array elements into a GDS structure, then extracting various layers from a `Struct` and storing them into `GArea` objects. The stored flexure and circle array GDS areas are then used to construct a perforated structure via boolean subtraction.

Lines 17 and 19 create two GDS structures. Line 21 stores the `flexure2C` element into an `ArrayList<GArea>`. Using the `createStruct` method, the vector array `al` is stored into a GDS structure (line 24). Line 27 stores an array of circles with GDS layer 7 into a GDS structure. Figure 4.362a shows the GDS rendered output.

Lines 30 and 32 extract GDS areas with particular GDS layer numbers and stores the shapes into the instantiated `GArea` objects. Line 34 performs a boolean subtraction between the flexure and the circle array. The resulting perforated area is stored into the `top` structure (line 37). Line 40 extracts the anchors and stores them into the `top` structure. Figure 4.362b shows the GDS rendered output.

![Figure 4.362](image)

**Figure 4.362:** Perforated flexures example showing GDS structures (a) flexureWithCircles and (b) top.

```java
package cnstprogrammingexamples;
import CNST.Scripting; import JGDS2.*;
import java.io.File;
import java.util.ArrayList;
```
public class Example05MEMSperforatedFlexure {
    public static void main(String[] args) {
        Lib lib = new Lib();
        // Struct top = new Struct("top");
        // Struct flexureWithCircles = new Struct("flexureWithCircles");
        // creating element flexure2C, structural layer = 4, anchor layer = 1
        ArrayList<GArea> al = Scripting.createFlexure2C(0, 0, 0.5, 5, 30, 2, 40, 20, 8, 10, 5.2, 4, 0.4, 1, 0, 4);

        // store ArrayList<GArea> into Struct
        Scripting.createStruct(flexureWithCircles, al);

        // creating an array of circles (circle layer = 7) in flexureWithCircles Struct
        Scripting.createSqrHoleC("circleArray", flexureWithCircles, 0, 0, 1, 1, 16, 4, 4, 10, 5, 0, 7);

        // extracting all objects with flexure layer 4
        GArea flexure = new GArea(flexureWithCircles.getArea(4), 4);
        // extracting all objects with circle layer 7
        GArea circle = new GArea(flexureWithCircles.getArea(7), 7);

        // boolean subtraction to create perforated flexure
        flexure.subtract(circle);

        // store boolean structure
        top.add(flexure);

        // extract and store anchors
        top.add(new GArea(flexureWithCircles.getArea(1), 1));
        //
        lib.add(new Ref(top, 0, 0));
        lib.add(new Ref(flexureWithCircles, 0, 0));

        // File f = lib.GDSOut("Example05MEMSperforatedFlexure.gds");
        System.out.println("Saved to " + f.getAbsolutePath());
    }
}
4.1.6 Custom Class Methods

The following example creates three methods identical to ones that generate yBend, yBendInv and yBendInvSlot structures described in corresponding sections 2.9.3.10, 2.9.3.11 and 2.9.3.12. Figure 4.363 shows the three 90 degree y-bend structures.

![Figure 4.363: Programming example of creating 90 degree (a) y-bend, (b) y-bend inverse and (c) y-bend inverse slot structures.](image)

The following code has four methods, main, createYBend90, createYBendInv90 and createYBendInvSlot90. Within the main method, line 14 initializes the GDS library and lines 16 to 20 initialize variables used to define the 90 degree y-bend structures. Lines 22 to 24 create GDS structures to store shapes created by the respective methods. Here we use the following constructor

```java
Struct(String structName, GDS2Element shape)
```

Line 22 creates a 90 degree y-bend (figure 4.363a) by implementing the createYBend90 method defined by lines 37-41. The method simply creates upper arc using the Scripting.createTorusW method (line 38). The GDS area undergoes a Boolean OR with the lower arc. The arc shape is defined by the constructor `torusW` in section 2.3.15.

Similarly, line 23 and 24 creates the remaining two 90 degree y-bend structures (figure 4.363b and 4.363c). The methods createYBendInv90 and createYBendInvSlot90 reuse the defined createYBend90 method to create the resulting shapes. For instance, createYBendInv90 first creates a 90 degree y-bend of width $w + 2w_e$ (line 47) and then subtracts a similar structure of width $w$ (line 48) to create a 90 degree y-bend inverse GDS area. Analogously, lines 54-58 define a method that creates a 90 degree y-bend inverse slot structure.
package cnstprogrammingexamples;

import CNST.Scripting;
import JCDS2.*;
import java.io.File;

/**
 * @author rob
 */

public class Example06CustomClassMethods {

    public static void main(String[] args) {
        Lib lib = new Lib();
        // define variables
        double x = 0, y = 0;
        double r1 = 20, r2 = 20;
        double w = 1.4, we = 1, ws = 1, g = 0.25;
        int numSides = 44, gdsLayer = 4;
        double THETA = 0;
        //
        Struct yBend90 = new Struct("yBend90Test", createYBend90(x, y,
            r1, r2, w, numSides, THETA, gdsLayer));
        Struct yBendInv90 = new Struct("yBendInv90Test",
            createYBendInv90(x, y, r1, r2, w, we, numSides, THETA,
            gdsLayer));
        Struct yBendInvSlot90 = new Struct("yBendInvSlot90Test",
            createYBendInvSlot90(x, y, r1, r2, ws, g, we, numSides,
            THETA, gdsLayer));
        //
        lib.add(new Ref(yBend90, 0, 0));
        lib.add(new Ref(yBendInv90, 0, 0));
        lib.add(new Ref(yBendInvSlot90, 0, 0));
        //
        File f = lib.GDSOut("Example06CustomClassMethods.gds");
        System.out.println("Saved to " + f.getAbsolutePath());
    }

    // yBend90
    // public static GArea createYBend90(double x, double y, double r1,
    // double r2, double w, int numSides, double THETA, int gdsLayer) {
    //     GArea ga = new GArea(Scripting.createTorusW(0, r1, r1, w, 270,
    // 360, numSides, 0, gdsLayer));
    //     ga.or(Scripting.createTorusW(0, -r2, r2, w, 0, 90, numSides, 0,
    // gdsLayer));
    //     return Scripting.transformGArea(ga, THETA, x, y);
    // }

    // yBendInv90
    // public static GArea createYBendInv90(double x, double y, double r1,
    // double r2, double w, double we, int numSides, double THETA,
    // int gdsLayer) {
    // }

}
GArea ga = createYBend90(x, y, r1, r2, w + 2 * we, numSides, THETA, gdsLayer);
return ga.subtract(createYBend90(x, y, r1, r2, w, numSides, THETA, gdsLayer));
}

// yBendInvSlot90
public static GArea createYBendInvSlot90(double x, double y, double r1, double r2, double ws, double g, double we, int numSides, double THETA, int gdsLayer) {
GArea ga = createYBend90(x, y, r1, r2, ws + 2 * we + 2 * g, numSides, THETA, gdsLayer);
ga.subtract(createYBend90(x, y, r1, r2, ws + 2 * g, numSides, THETA, gdsLayer));
return ga.or(createYBend90(x, y, r1, r2, ws, numSides, THETA, gdsLayer));
}

4.1.7 GDS Area Objects - Layers and Data-types

GDS layer definitions are typically defined within the method constructor. Layer data-types are either defined globally for subsequent GDS objects or GDS area objects are assigned to a data-type. For instance, within the following 4 lines, a structure is created, layer data-type 44 is set for subsequent shapes, the GDS area ellipse acquires the data-type value, and is added to the Struct.

1 Struct top = new Struct("top");
2 Scripting.setDataType(44);
3 GArea ellipse = Scripting.createEllipse(0, 0.4, 0.8, 44, 0, 4);
4 top.add(ellipse);

The following is an equivalent strategy that casts the GDS area object onto a particular datatyp.
4.1.8 Centering GDS Area Objects

The following code will find a centroid of a GDS shape. This procedure is useful for centering features around a specified origin or to create symmetric pattern extents. Lines 18 and 19 define a GDS structure and area, respectively. The GDS area is then added to the structure (line 20). In line 22, a new structure is formed that will store the resulting centered GDS area. The GDS area is extracted and stored into a Java Area (line 23). Using the built-in Java Rectangle2D class, the area boundary is extracted in line 24. The GDS structure containing the GDS area is then instantiated with offsets determined by the getCenterX() and getCenterY() methods from the Rectangle2D class. Instead of instantiating, the GDS area object could be added directly to a GDS structure using affine translational transforms with (-rec.getCenterX(), -rec.getCenterY()).

```java
package cnstprogrammingexamples;
import CNST.Scripting;
import JGDS2.*;
import java.awt.geom.Area;
import java.awt.geom.Rectangle2D;
import java.io.File;

/*
 * @author rob
 */
public class Example07AreaExtents {
  public static void main(String[] args) {
    Lib lib = new Lib();
    // creating and storing a y-bend
    Struct yBendTest = new Struct("yBendExample");
    GArea yBend = Scripting.createYbend(0, 0, 10, 10, 20, -40, 1, 0, 7, 0.01);
    yBendTest.add(yBend);
    // extracting pattern extents
    Struct yBendCentered = new Struct("yBendCenteredShape");
    Area a = new Area(yBend.getArea());
    Rectangle2D rec = a.getBounds2D();
    yBendCentered.add(new Ref(yBendTest, -rec.getCenterX(), -rec.getCenterY()));
    //
    lib.add(new Ref(yBendTest, 0, 0));
    lib.add(new Ref(yBendCentered, 0, 0));
    File f = lib.GDSOut("Example07AreaExtents.gds");
    System.out.println("Saved to " + f.getAbsolutePath());
  }
}
```
4.1.9 PostScript to GDS

The below code creates a GDS area using a string containing postscript commands. Unlike scripting, where the number of postscript commands is practically unlimited, with the programming interface it is limited. This limitation stems from the Java String length limit. As defined by the Java specifications, a String has a maximum character length of \( \text{Integer.MAX VALUE} = (2^{31} - 1) \).

A host of memory dilemmas will be encountered prior reaching this limit, hence instead of holding all postscript commands in one String, the commands could be distributed over multiple Strings.

Example 8 implements an interface (Example08PSInterface.java) that contains a string \( s \) with postscript paths (line 11). Variables are defined and a GDS structure is created (lines 16 to 21) in a similar manner to previous programming examples. Line 22 uses a method \texttt{createPostScript} in conjunction with a String \texttt{psString} to generate output GDS shapes. String variable \texttt{psString} is defined in the Example08PSInterface.java interface file. The following two methods are available for creating GDS shapes from vectorized objects:

```java
public static ArrayList<GArea> createPostScript(double x, double y,
        String s, int fractureSegments, double THETA, double shapeReso,
        double pixelValue, int gdsLayer)

public static void createPostScript(Struct currentStruct, double x,
        double y, String s, int fractureSegments, double THETA, double
        shapeReso, double pixelValue, int gdsLayer)
```

In our example we used the \texttt{void} method (line 22), that directly stores generated shapes into a specified (currentStruct) GDS structure. The lower left corner of the structure is positioned at \((x, y)\). The resulting structure is fractured into a number of equal segments defined by the integer value of \texttt{fractureSegments}. Parameters \texttt{THETA}, \texttt{shapeReso}, \texttt{pixelValue} and \texttt{gdsLayer} are rotation about \((x, y)\), rendering resolution of the vectorized shape (section 2.2.6), pixel scaling value of the postscript coordinates (section 2.6.5.1) and GDS layer number, respectively.

1 package cnstprogrammingexamples;
2
3 import CNST.Scripting;
4 import JGDS2.*;
5 import java.io.File;
6
7 //
8 * @author rob
9 */
10
11 public class Example08PostScriptFracturing implements Example08PSInterface {
```java
public static void main(String[] args) {
    Lib lib = new Lib();
    // variables
    int gdsLayer = 44;
    double shapeReso = 0.001;
    double pixelValue = .1;
    int fractureSegments = 10;
    //
    Struct ps2gds = new Struct("postScript2GDS");
    Scripting.createPostScript(ps2gds, 0, 0, psString,
        fractureSegments, 0, shapeReso, pixelValue, gdsLayer);
    //
    lib.add(new Ref(ps2gds, 0, 0));
    File f = lib.GDSOut("Example08PostScriptFracturing.gds");
    System.out.println("Saved to " + f.getAbsolutePath());
}
```
4.1.10 Curved Fluidic Channels

The following example creates two sets of labeled fluidic channels using predefined scripting methods. Straight segments were constructed using the waveguide method that defines a rectangular shape between points \((x_1, y_1)\) and \((x_2, y_2)\). The curved segments were created using either bezier curves or 90 degree torus bends.

Figure 4.364: Programming example of labeled fluidic channels.
// layerChannel = 4, layerChannel2 = 7, numElectrodes = 8;
double shapeReso = 0.1, fontSize = 250;
double chipSize = 10000, electrodeRadius = 250,
electrodeSpacing = 100, channelWidth = 20;
GArea electrode = Scripting.createEllipse(0, 0, electrodeRadius,
electrodeRadius, 100, 0, layerChannel);
Struct chip = new Struct("chip");
AffineTransform at = new AffineTransform(new double[]{−1.0, 0.0, 0.0, 1.0});
// side channels 1−8
for (int i = 0; i < numElectrodes; i++) {
  GArea channels = new GArea(electrode);
  GArea label = Scripting.createTextGdsC("" + (i + 1), "Serif",
    fontSize, −electrodeRadius − fontSize / 2, 0, 0,
    layerChannel, shapeReso);
  double x = 2 − electrodeRadius;
  double y = chipSize / 4 + i * (2 − electrodeRadius +
electrodeSpacing);
  double x2 = x + 200 − i * channelWidth + 2;
  double y3 = chipSize − 1500 − (numElectrodes − i
    − 1) * 40 − y;
  channels.or(Scripting.createWaveGuide(0, 0, x2, 0,
    channelWidth, 0, 0, 0, layerChannel));
  channels.or(Scripting.create90degreeBendLH(x2, 0, x3, 200,
    channelWidth, 0, layerChannel, shapeReso));
  channels.or(Scripting.createWaveGuide(x2 + x3, 200, x2 + x3
    , y3, channelWidth, 0, 0, 0, layerChannel));
  double radiusChannel = 1000;
  double x4 = x2 + x3 + radiusChannel;
  channels.or(Scripting.createTorusW(x4, y3, radiusChannel,
    channelWidth, 90, 180, 100, 0, layerChannel));
  double y4 = y3 + radiusChannel;
  double x5 = chipSize / 2 − 2000 − (numElectrodes − i − 1) * 40;
  channels.or(Scripting.createWaveGuide(x4, y4, x5, y4,
    channelWidth, 0, 0, 0, layerChannel));
  radiusChannel = 500;
  double y5 = y4 − radiusChannel;
  channels.or(Scripting.createTorusW(x5, y5, radiusChannel,
    channelWidth, 90, 180, 100, 0, layerChannel));
  double x6 = x5 + radiusChannel;
  double y6 = −y + 2500 − (numElectrodes − i − 1) * 100;
  channels.or(Scripting.createWaveGuide(x6, y5, x6, y6,
    channelWidth, 0, 0, 0, layerChannel));
  radiusChannel = 500;
  double x7 = x6 + radiusChannel;
  channels.or(Scripting.createTorusW(x7, y6, radiusChannel,
    channelWidth, 180, 270, 100, 0, layerChannel));
  double y7 = y6 − radiusChannel;
  double x8 = chipSize / 2 − x;
  channels.or(Scripting.createWaveGuide(x7, y7, x8, y7,
    channelWidth, 0, 0, 0, layerChannel));
  channels.transform(AffineTransform.getTranslateInstance(−
    chipSize / 2 + x, −chipSize / 2 + y));
  GArea channelsMirror = new GArea(channels);
  channelsMirror.transform(at); // mirror around y-axis
chip.add(channels);
chip.add(channelsMirror);
chip.add(label.transform(AffineTransform.
getTranslateInstance(-chipSize / 2 + x, -chipSize / 2 + y)));
GArea temp = new GArea(label);
chip.add(temp.transform(AffineTransform.
getTranslateInstance(chipSize - x / 2, 0)));
]
// bottom channels
electrodeSpacing = 200;
channelWidth = 50;
double offsetX = 400;
for (int i = 0; i < numElectrodes / 2; i++) {
  GArea channels = new GArea(electrode);
double x = (i + 1) * (2 * electrodeRadius + electrodeSpacing) + offsetX;
double y = 2.5 * electrodeRadius;
double x2 = x - i * channelId + 20;
double y2 = 0.55 * chipSize - y - i * channelId + 4;
channels.or(Scripting.createBezierCurve(x, electrodeRadius
  / 2, 0, channelWidth / 4, x2, chipSize / 8, x2, y2,
  channelId, shapeReso, 0, layerChannel2));
double radiusChannel = 400;
double x3 = x2 + radiusChannel;
channels.or(Scripting.createTorusW(x3, y2, radiusChannel,
  channelId, 90, 180, 100, 0, layerChannel2));
double y3 = y2 + radiusChannel;
double x4 = chipSize / 2 - x;
channels.or(Scripting.createWaveGuide(x3, y3, x4, y3,
  channelId, 0, 0, 0, layerChannel));
channels.transform(AffineTransform.
getTranslateInstance(-chipSize / 2 + x, -chipSize / 2 + y));
channels.setLayer(layerChannel2);
GArea channelsMirror = new GArea(channels);
channelsMirror.transform(at); // mirror around y-axis
chip.add(channels);
chip.add(channelsMirror);
]
// Labels for the bottom electrodes
double x = -chipSize / 2 + (2 * electrodeRadius + electrodeSpacing) + offsetX - fontSize / 4;
double y = -chipSize / 2 + fontSize / 2;
double pitch = (2 * electrodeRadius + electrodeSpacing);
Scripting.createLabelMakerAutoOutLett(0, 4, "Serif", fontSize,
  x, y, 0, 0, pitch, 1, chip,
  layerChannel2, shapeReso); // label A B C D
x = Math.abs(x) - fontSize / 2;
Scripting.createLabelMakerAutoOutLett(0, 4, "Serif", fontSize,
  x, y, 0, 0, -pitch, 1, chip, layerChannel2, shapeReso); // label D C B A
]
// lib.add(new Ref(chip, 0, 0));
File f = lib.GDSOut("fluidics.gds");
System.out.println("Saved to " + f.getAbsolutePath());
}
4.2 Reference Methods

The following subsection contain method headers that create complex GDS objects. These shapes are fully described in the GDS scripting chapter. Excluding access, mutator and miscellaneous methods, the shape methods names have the create followed by the script constructor name with the first letter capitalized. For instance, the ellipse scripting constructor (also shown in the previous comment line) would have a method entitled `createEllipse`

```java
// ellipse
public static GArea createEllipse(double x, double y, double radiusX,
    double radiusY, int numSides, double THETA, int gdsLayer)
```

Directly following the method name are a set of parameters. All methods have a parameter list with the exception of accessor methods that have empty parameter lists. Methods return types are either `void`, `GArea` or `ArrayList<Area>`. Since `Struct` objects add `GArea` objects but not array lists of these objects, each vector array return types also have a void method. The void method stores the `ArrayList<Area>` into the passed `Struct currentStruct`. Section 4.2.2 contains methods for dealing with `GArea` or `ArrayList<Area>` objects. These are useful for controlling GDS area objects in custom user defined methods.

4.2.1 Accessor and Mutator Methods

```java
public static double getFontOutline()
public static int getFracElements()
public static double getGDSreso()
public static int getLayer()
public static int getDataType()
public static double getPixelValue()
public static double getShapeReso()
public static void setFontOutline(double fontOutlineWidth)
public static void setFracElements(int fracElements)
public static void setGDSreso(double gdsReso)
public static void setLayer(int layer)
public static void setDataType(int layer)
public static void setPixelValue(double pixelValue)
public static void setShapeReso(double shapeReso)
```
4.2.2 Miscellaneous GDS Area and Struct Methods

These methods are used for affine transformations of single and vector array GDS area objects. Furthermore, the void methods place constructed vector arrays of GArea objects into a GDS structure.

```java
// createStruct - place ga into currentStruct
public static void createStruct(Struct currentStruct, ArrayList<GArea> ga)

// createStruct - mirror ga then place ga into currentStruct
public static void createStruct(Struct currentStruct, ArrayList<GArea> ga, boolean MIRROR)

// createStructRotate - rotation only - then place ga into currentStruct
public static void createStructRotate(Struct currentStruct, ArrayList<GArea> ga, double THETA, double x, double y)

// createStructTranslateRotate - translate then rotate ga then place ga into currentStruct
public static void createStructTranslateRotate(Struct currentStruct, ArrayList<GArea> ga, double THETA, double x, double y)

// rotateGArea - rotate GArea v about a point (x,y)
public static GArea rotateGArea(GArea v, double THETA, double x, double y)

// transformGArea - translate v to (x,y) then rotate about (x,y)
public static GArea transformGArea(GArea v, double THETA, double x, double y)

// rotateArrayListGArea - rotate ArrayList ga about (x,y)
public static ArrayList<GArea> rotateArrayListGArea(ArrayList<GArea> ga, double THETA, double x, double y)

// transformArrayListGArea - ArrayList ga translate to (x,y) then rotate about (x,y)
public static ArrayList<GArea> transformArrayListGArea(ArrayList<GArea> ga, double THETA, double x, double y)
```
4.2.3 Shape Methods

// ellipse
public static GArea createEllipse(double x, double y, double radiusX, double radiusY, int numSides, double THETA, int gdsLayer)

// ellipseVector
public static GArea createEllipseVector(double x, double y, double radiusX, double radiusY, double THETA, int gdsLayer, double shapeReso)

// circleThree
public static GArea createCircleThree(double x1, double y1, double x2, double y2, double x3, double y3, int numSides, double THETA, int gdsLayer)

// circleWave
public static GArea createCircleWave(double x, double y, double r, int n, double A, int numSides, double THETA, int gdsLayer)

// cross
public static GArea createCross(double x, double y, double W, double L, double THETA, int gdsLayer)

// Lshape
public static GArea createLshape(double x, double y, double W1, double L1, double W2, double L2, double THETA, int gdsLayer)

// arc
public static GArea createArc(double x, double y, double rX, double rY, double angleStart, double angleEnd, int numSides, double THETA, int gdsLayer)

// arcVector
public static GArea createArcVector(double x, double y, double rX, double rY, double angleStart, double angleEnd, double THETA, int gdsLayer, double shapeReso)

// polygon
public static GArea createPolygon(double x, double y, double r, int numSides, double THETA, int gdsLayer, double shapeReso)

// rectangle
public static GArea createRectangle(double xLL, double yLL, double xUR, double yUR, double THETA, int gdsLayer)
// rectangleLH
public static GArea createRectangleLH(double xLL, double yLL, double L,
    double H, double THETA, int gdsLayer)

// rectangleC
public static GArea createRectangleC(double xC, double yC, double L,
    double H, double THETA, int gdsLayer)

// roundrect
public static GArea createRoundRect(double xLL, double yLL, double L,
    double H, double rX, double rY, double THETA, int gdsLayer, double shapeReso)

// rectSUshape
public static GArea createRectSUshape(double x, double y, double L1,
    double L2, double L3, double W, double THETA, int gdsLayer)

// rectTaper
public static GArea createRectTaper(double x, double y, double w1,
    double L1, double w2, double L2, double THETA, int gdsLayer)

// star
public static GArea createStar(double x, double y, double rIn, double rOut,
    int numPoints, double THETA, int gdsLayer, double shapeReso)

// torus
public static GArea createTorus(double x, double y, double rIn, double rOut,
    double angleStart, double angleEnd, int numPoints, double THETA, int gdsLayer)

// torusW
public static GArea createTorusW(double x, double y, double r, double width,
    double angleStart, double angleEnd, int numPoints, double THETA, int gdsLayer)

// torusVector
public static GArea createTorusVector(double x, double y, double rIn,
    double rOut, int gdsLayer, double shapeReso)

// torusWaveIn
public static GArea createTorusWaveIn(double x, double y, double rIn,
    double rOut, int n, double A, int numSides, double THETA, int gdsLayer)

// torusWaveOut
public static GArea createTorusWaveOut(double x, double y, double rIn,
    double rOut, int n, double A, int numSides, double THETA, int gdsLayer)
4.2.4 Array Methods

// arrayRectV1 public static void createArrayRectV1(Struct structToBeArrayed, Struct currentStruct, double x, double y, int numColumns, int numRows, double dx, double dy)

// arrayRectV2 public static void createArrayRectV2(Struct structToBeArrayed, Struct currentStruct, double x, double y, int numColumns, int numRows, double xE, double yE)

// arrayHex public static void createArrayHex(Struct structToBeArrayed, Struct currentStruct, double x, double y, int numColumns, int numRows, double ds)

// arrayPolarV1 public static void createArrayPolarV1(Struct structToBeArrayed, Struct currentStruct, double angleStart, double angleEnd, double deltaAngle, double radiusStart, double radiusEnd, double deltaRadius)

// arrayPolarV1R public static void createArrayPolarV1R(Struct structToBeArrayed, Struct currentStruct, double angleStart, double angleEnd, double deltaAngle, double radiusStart, double radiusEnd, double deltaRadius)

// arrayPolarV2 public static void createArrayPolarV2(Struct structToBeArrayed, Struct currentStruct, double angleStart, double angleEnd, double numberOfAngles, double radiusStart, double radiusEnd, double numberOfRadii)

// arrayPolarV2R public static void createArrayPolarV2R(Struct structToBeArrayed, Struct currentStruct, double angleStart, double angleEnd, double numberOfAngles, double radiusStart, double radiusEnd, double numberOfRadii)

Instantiation using instance, instanceSym and points2instance can be accomplished using the JGDS reference (Ref) constructor. Refer to the JGDS manual for information on Ref.
// sqrPillar
public static void createSqrPillar(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer)

// hexPillar
public static void createHexPillar(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, int numElementsX, int numElementsY, double THETA, int gdsLayer)

// sqrHole
public static void createSqrHole(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer)

// hexHole
public static void createHexHole(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, int numElementsX, int numElementsY, double THETA, int gdsLayer)

// sqrPillarC
public static void createSqrPillarC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer)

// hexPillarC
public static void createHexPillarC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, int numElementsX, int numElementsY, double THETA, int gdsLayer)

// sqrHoleC
public static void createSqrHoleC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer)

// hexHoleC
public static void createHexHoleC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, int numSides, double pitchX, int numElementsX, int numElementsY, double THETA, int gdsLayer)
// s q r P i l l a r V
public static void createSqrPillarV(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)

// h e x P i l l a r V
public static void createHexPillarV(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)

// s q r H o l e V
public static void createSqrHoleV(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)

// h e x H o l e V
public static void createHexHoleV(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)

// s q r P i l l a r V C
public static void createSqrPillarVC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)

// h e x P i l l a r V C
public static void createHexPillarVC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)

// s q r H o l e V C
public static void createSqrHoleVC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, double pitchY, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)

// h e x H o l e V C
public static void createHexHoleVC(String uniqueStructName, Struct currentStruct, double x, double y, double rX, double rY, double pitchX, int numElementsX, int numElementsY, double THETA, int gdsLayer, double shapeReso)
4.2.5 Text, Labels, PostScript and Logo Methods

// textgds
class public static GArea createTextGds(String textString, String fontName,
double fontSize, double x, double y, double THETA, int gdsLayer,
double shapeReso)

// textgdsC
class public static GArea createTextGdsC(String textString, String fontName,
double fontSize, double x, double y, double THETA, int gdsLayer,
double shapeReso)

// textOutline
class public static ArrayList<Area> createTextOutline(String textString,
String fontName, double fontSize, double x, double y, double THETA,
double fontOutlineWidth, int gdsLayer, double shapeReso)

// textOutline
class public static void createTextOutline(String textString, String fontName,
double fontSize, double x, double y, double fontOutlineWidth, int gdsLayer, double shapeReso)

// textOutlineC
class public static ArrayList<Area> createTextOutlineC(String textString,
String fontName, double fontSize, double x, double y, double THETA,
double fontOutlineWidth, int gdsLayer, double shapeReso)

// textOutlineC
class public static void createTextOutlineC(String textString, String fontName,
double fontSize, double x, double y, double fontOutlineWidth, int gdsLayer, double shapeReso)

// labelMaker autoOut
class public static void createLabelMakerAutoOut(int rows, int columns,
String fontName, double fontSize, double x, double y, double xRow,
double yRow, double pitchX, double pitchY, Struct currentStruct,
int gdsLayer, double shapeReso)

// labelMaker autoRowCol
class public static void createLabelMakerAutoRowCol(int rows, int columns,
String fontName, double fontSize, double x, double y, double pitchX
, double pitchY, Struct currentStruct, int gdsLayer, double shapeReso)

// labelMaker autoOutLett
class public static void createLabelMakerAutoOutLett(int rows, int columns,
String fontName, double fontSize, double x, double y, double xRow,
double yRow, double pitchX, double pitchY, Struct currentStruct,
int gdsLayer, double shapeReso)

// labelMaker autoRowColLett
public static void createLabelMakerAutoRowColLett(int rows, int columns,
String fontName, double fontSize, double x, double y, double pitchX, double pitchY,
Struct currentStruct, int gdsLayer, double shapeReso)

// labelMaker CUSTOM OUTER
public static void createLabelMakerCustomOuter(String[] stringArray,
int rows, int columns, String fontName, double fontSize, double x,
double y, double xRow, double yRow, double pitchX, double pitchY,
Struct str, int gdsLayer, double shapeReso)

// labelMaker CUSTOM ROW COLUMN label
public static void createLabelMakerCustomRowColumn(String[] stringArray,
int rows, int columns, String fontName, double fontSize, double x,
double y, double pitchX, double pitchY, Struct str, int gdsLayer,
double shapeReso)

// labelOutline autoOut
public static void createLabelOutlineAutoOut(int rows, int columns,
String fontName, double fontSize, double x, double y, double xRow,
double yRow, double pitchX, double pitchY, Struct currentStruct,
double fontOutlineWidth, int gdsLayer, double shapeReso)

// labelOutline autoRowCol
public static void createLabelOutlineAutoRowCol(int rows, int columns,
String fontName, double fontSize, double x, double y, double pitchX,
double pitchY, Struct currentStruct, double fontOutlineWidth, int
shapeReso)

// labelOutline autoRowColLett
public static void createLabelOutlineAutoRowColLett(int rows, int columns,
String fontName, double fontSize, double x, double y, double pitchX,
double pitchY, Struct currentStruct, double fontOutlineWidth, int
gdsLayer, double shapeReso)

// labelOutline CUSTOM OUTER
public static void createLabelOutlineCustomOuter(String[] stringArray,
int rows, int columns, String fontName, double fontSize, double x,
double y, double xRow, double yRow, double pitchX, double pitchY,
Struct str, double fontOutlineWidth, int gdsLayer, double shapeReso
)

// labelOutline CUSTOM ROW COLUMN label
public static void createLabelOutlineCustomRowColumn(String[] stringArray, int rows, int columns, String fontName, double
fontSize, double x, double y, double pitchX, double pitchY, Struct str, double fontOutlineWidth, int gdsLayer, double shapeReso)

// postScript
public static ArrayList<GArea> createPostScript(double x, double y,
    String s, int fractureSegments, double THETA, double shapeReso,
    double pixelValue, int gdsLayer)

// postScript
public static void createPostScript(Struct currentStruct, double x,
    double y, String s, int fractureSegments, double THETA, double
    shapeReso, double pixelValue, int gdsLayer)

// cnstEmblem
public static GArea createCnstEmblemLogo(double x, double y, double
    scale, double THETA, double shapeReso, int gdsLayer) throws
    FileNotFoundException

// cnstLogo
public static GArea createCnstLogo(double x, double y, double scale,
    double THETA, double shapeReso, int gdsLayer) throws
    FileNotFoundException

// nistLogo
public static GArea createNistLogo(double x, double y, double scale,
    double THETA, double shapeReso, int gdsLayer) throws
    FileNotFoundException

// nistCnstLogo
public static GArea getNistCnstLogo(double x, double y, double scale,
    double THETA, double shapeReso, int gdsLayer) throws
    FileNotFoundException
4.2.6 Miscellaneous Object Methods

```java
// arcSquareFill
public static void createArcSquareFill(Struct currentStruct, String uniqueStructName, double x, double y, double rIn, double rOut, double angleStart, double angleEnd, int numSides, double THETA, double rSmallObject, int numSidesSmallObject, double pitchX, int gdsLayer)

// arcSquareFillV
public static void createArcSquareFillV(Struct currentStruct, String uniqueStructName, double x, double y, double rIn, double rOut, double angleStart, double angleEnd, int numSides, double THETA, double rSmallObject, double pitchX, double shapeReso, int gdsLayer)

// arcHexFill
public static void createArcHexFill(Struct currentStruct, String uniqueStructName, double x, double y, double rIn, double rOut, double angleStart, double angleEnd, int numSides, double THETA, double rSmallObject, int numSidesSmallObject, double pitchX, int gdsLayer)

// arcSquareFillV
public static void createArcHexFillV(Struct currentStruct, String uniqueStructName, double x, double y, double rIn, double rOut, double angleStart, double angleEnd, int numSides, double THETA, double rSmallObject, double pitchX, double shapeReso, int gdsLayer)

// bezierCurve
public static GArea createBezierCurve(double x1, double y1, double cx1, double cy1, double cx2, double cy2, double x2, double y2, double width, double shapeReso, double THETA, int gdsLayer)

// bezierCurveln
public static GArea createBezierCurveln(double x1, double y1, double cx1, double cy1, double cx2, double cy2, double x2, double y2, double sleeveWidth, double slotWidth, double shapeReso, double THETA, int gdsLayer)

// bezierCurvelnSlot
public static GArea createBezierCurvelnSlot(double x1, double y1, double cx1, double cy1, double cx2, double cy2, double x2, double y2, double sleeveWidth, double slotWidth, double gap, double shapeReso, double THETA, int gdsLayer)

// sierpinskiTriangle
public static void createSierpinskiTriangle(Struct currentStruct, String shortStructName, double x, double y, int iterations, double length, int gdsLayer)
```
// sierpinskiCarpet
public static void createSierpinskiCarpet(Struct currentStruct, String shortStructName, double x, double y, int iterations, double length, int gdsLayer)

// vicsekSaltire
public static void createVicsekSaltire(Struct currentStruct, String shortStructName, double x, double y, int iterations, double length, int gdsLayer)

// vicsekCross
public static void createVicsekCross(Struct currentStruct, String shortStructName, double x, double y, int iterations, double length, int gdsLayer)

// curvedTree
public static void createCurvedTree(Struct currentStruct, String shortStructName, double x, double y, double length, double width, int iterations, int gdsLayer)

// anotherTree fractal - undocumented Fractal
public static void createAnotherTree(Struct currentStruct, String shortStructName, double x, double y, int iterations, int gdsLayer)

// complex tree fractal - undocumented Fractal - vary parameters
public static void createTree(Struct currentStruct, String shortStructName, double x, double y, double d, double ang, double ratio, int depth, int gdsLayer)

// functionXY
public static ArrayList<GArea> createFunctionXY(String function, double xLowerLimit, double xUpperLimit, int N, double W, int CAP, int JOIN, int gdsLayer) throws ScriptException

// functionXY
public static void createFunctionXY(Struct currentStruct, String function, double x, double y, double xLowerLimit, double xUpperLimit, int N, double W, int CAP, int JOIN, double THETA) throws ScriptException

// functionRTheta
public static ArrayList<GArea> createFunctionRTheta(String function, double x, double y, double thetaLowerLimit, double thetaUpperLimit, int N, double W, int CAP, int JOIN, int gdsLayer) throws ScriptException

// functionRTheta
public static void createFunctionRTheta(Struct currentStruct, String function, double x, double y, double thetaLowerLimit, double thetaUpperLimit, int N, double W, int CAP, int JOIN, int gdsLayer) throws ScriptException
// functionRTheta
public static void createFunctionRTheta(Struct currentStruct, String function, double x, double y, double thetaLowerLimit, double thetaUpperLimit, int N, double W, int CAP, int JOIN, double THETA) throws ScriptException

// grayERamp
public static ArrayList<Area> createGrayERamp(double x, double y, double dx, double dy, int numberOfSegments, double THETA)

// grayENgon
public static ArrayList<Area> createGrayENgon(double x, double y, double radX, double radY, int numberOfSegments, int numSides, double THETA)

// grayUDR - al contains dx and dy coordinate pairs
public static ArrayList<Area> createGrayUDR(double x, double y, ArrayList<Double> al, double THETA)

// grayUDNgon - al contains dx and dy coordinate pairs
public static ArrayList<Area> createGrayUDNgon(double x, double y, ArrayList<Double> al, int numSides, double THETA)

// grayERamp - UP = true or false
public static ArrayList<Area> createGrayERamp(double x, double y, double dx, double dy, int numberOfSegments, boolean UP, double THETA)
public static void createGrayERamp(Struct currentStruct, double x, double y, double dX, double dY, int numberOfSegments, boolean UP, double THETA)

// grayERamp
public static ArrayList<GArea> createGrayERamp2(double x, double y, double dX, double dY, int numberOfSegments, double THETA)

// grayERamp2
public static void createGrayERamp2(Struct currentStruct, double x, double y, double dX, double dY, int numberOfSegments, double THETA)

// grayUDRamp - al contains x1, x2, ..., xn
public static ArrayList<GArea> createGrayUDRamp(double x, double y, ArrayList<Double> al, double dY, boolean UP, double THETA)

// grayUDRamp2 - al contains x1, x2, ..., xn
public static void createGrayUDRamp2(Struct currentStruct, double x, double y, ArrayList<Double> al, double dY, boolean UP, double THETA)

// grayUDRamp2 - al contains x1, x2, ..., xn
public static ArrayList<GArea> createGrayUDRamp2(double x, double y, ArrayList<Double> al, double dY, double THETA)

// grayUDRamp2 - al contains x1, x2, ..., xn
public static void createGrayUDRamp2(Struct currentStruct, double x, double y, ArrayList<Double> al, double dY, double THETA)

// graySpiralStairOverlap - create grayscale OVERLAPPING ARCs
public static void createGraySpiralStairOverlap(Struct currentStruct, double x, double y, double rIn, double rOut, int numSegments, int numSides, double THETA)

// graySpiralStair - create grayscale NON-Overlapping ARCs
public static ArrayList<GArea> createSpiralStair(double x, double y, double rIn, double rOut, int numSegments, int numSides, double THETA)

// graySpiralStair - create grayscale NON-Overlapping ARCs
public static void createGraySpiralStair(Struct currentStruct, double x, double y, double rIn, double rOut, int numSegments, int numSides, double THETA)

// intElec1
public static ArrayList<GArea> createIntElec1(double x, double y, double width1, double width2, double length1, double length2, double overlap, int numElectrodes, double pitch, double baseHeight,
double baseWidth, double THETA, int gdsLayer)

// intElec1
public static void createIntElec1(Struct currentStruct, double x,
    double y, double width1, double width2, double length1, double
    length2, double overlap, int numElectrodes, double pitch, double
    baseHeight, double baseWidth, double THETA, int gdsLayer)

// intElec2
public static ArrayList<GArea> createIntElec2(double x, double y,
    double width1, double width2, double length1, double length2,
    double overlap, int numElectrodes, double pitch, double
    baseHeight, double baseWidth, double THETA, int gdsLayer)

// intElec3
public static ArrayList<GArea> createIntElec3(double x, double y,
    double width1, double width2, double length1, double length2,
    double overlap, int numElectrodes, double pitch, double
    baseHeight, double baseWidth, double THETA, int gdsLayer)

// intElec4
public static ArrayList<GArea> createIntElec4(double x, double y,
    double width1, double width2, double length1, double length2,
    double overlap, int numElectrodes, double pitch, double
    baseHeight, double baseWidth, double THETA, int gdsLayer)

// intElec5
public static ArrayList<GArea> createIntElec5(double x, double y,
    double width1, double width2, double length1, double length2,
    double overlap, int numElectrodes, double pitch, double
    baseHeight, double baseWidth, double THETA, int gdsLayer)
// intElec5
public static void createIntElec5(Struct currentStruct, double x, double y, double width1, double width2, double length1, double length2, double overlap, int numElectrodes, double pitch, double baseHeight, double baseWidth, double THETA, int gdsLayer)

// tJunction
public static ArrayList<GArea> createTjunction(double x, double y, double w1, double w2, double L1, double L2, double rad, int numSides, double THETA, int gdsLayer)

// tJunction
public static ArrayList<GArea> createTjunction(Struct currentStruct, double x, double y, double w1, double w2, double L1, double L2, double rad, int numSides, double THETA, int gdsLayer)

// hJunction
public static ArrayList<GArea> createHjunction(double x, double y, double w1, double w2, double L1, double L2, double rad, int numSides, double THETA, int gdsLayer)

// hJunction
public static ArrayList<GArea> createHjunction(Struct currentStruct, double x, double y, double w1, double w2, double L1, double L2, double rad, int numSides, double THETA, int gdsLayer)

// arrowJunction
public static ArrayList<GArea> createArrowJunction(double x, double y, double w1, double w2, double L1, double L2, double rad, int numSides, double junctionAngle, double THETA, int gdsLayer)

// arrowJunction
public static ArrayList<GArea> createArrowJunction(Struct currentStruct, double x, double y, double w1, double w2, double L1, double L2, double rad, int numSides, double junctionAngle, double THETA, int gdsLayer)

// meanderSin
public static ArrayList<GArea> createMeanderSin(double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, int numCurveSegments, double a, double b, double c, double THETA, int gdsLayer)

// meanderSin
public static ArrayList<GArea> createMeanderSin(Struct currentStruct, double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, int numCurveSegments, double a, double b, double c, double THETA, int gdsLayer)
// meanderSqr
public static ArrayList<GArea> createMeanderSqr(double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, double a, double b, double c, double THETA, int gdsLayer)

// meanderSqr
public static void createMeanderSqr(Struct currentStruct, double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, double a, double b, double c, double THETA, int gdsLayer)

// meanderRamp
public static ArrayList<GArea> createMeanderRamp(double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, double a, double b, double c, double THETA, int gdsLayer)

// meanderRamp
public static void createMeanderRamp(Struct currentStruct, double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, double a, double b, double c, double THETA, int gdsLayer)

// meanderTri
public static ArrayList<GArea> createMeanderTri(double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, double a, double b, double c, double THETA, int gdsLayer)

// meanderTri
public static void createMeanderTri(Struct currentStruct, double x, double y, double L1, double H1, double L2, double H2, double width, double amplitude, int numPeriods, double a, double b, double c, double THETA, int gdsLayer)

// points2shape
public static GArea createPoints2Shape(double x, double y, ArrayList<Double> al, double THETA, int gdsLayer)

// polypath
public static GArea createPolyPath(double x, double y, ArrayList<Double> al, double width, int CAP, int JOIN, double THETA, int gdsLayer)

// randomPolygons
public static void createRandomPolygons(Struct currentStruct, String uniqueStructName, double x, double y, double width, double height,
public static void createRandomEllipses(Struct currentStruct, String uniqueStructName, double x, double y, double width, double height, double radiusX, double radiusY, int numSides, double separation, int numElements, int iterations, boolean RANDOMROTATION, int gdsLayer)

public static void createRandomEllipsesV(Struct currentStruct, String uniqueStructName, double x, double y, double width, double height, double radiusX, double radiusY, double separation, int numElements, int iterations, boolean RANDOMROTATION, int gdsLayer, double shapeReso)

public static ArrayList<GArea> createResoPattern(double x, double y, double rad, double width, double THETA, int gdsLayer)

public static void createResoPattern(Struct currentStruct, double x, double y, double rad, double width, double THETA, int gdsLayer)

public static ArrayList<GArea> createResoPatternPi(double x, double y, double rad, double width, int n, double THETA, int gdsLayer)

public static void createResoPatternPi(Struct currentStruct, double x, double y, double rad, double width, int n, double THETA, int gdsLayer)

public static GArea createResoPatternRS(double x, double y, double W, double H, int numberOfLines, double THETA, int gdsLayer, double shapeReso)

public static void createResoPatternRS(Struct currentStruct, double x, double y, double W, double H, int numberOfLines, double THETA, int gdsLayer, double shapeReso)

public static ArrayList<GArea> createResoPatternRSA(double x, double y, double startW, double endW, double delta, double H, int numberOfLines, double space, double THETA, int gdsLayer, double
shapeReso)

// resoPatternRSA
public static void createResoPatternRSA(Struct currentStruct, double x,
  double y, double startW, double endW, double delta, double H, int
  numberOfLines, double space, double THETA, int gdsLayer, double
  shapeReso)

// resoPatternLS
public static GArea createResoPatternLS(double x, double y, double W,
double H, int numberOfLines, double THETA, int gdsLayer, double
shapeReso)

// resoPatternLS
public static void createResoPatternLS(Struct currentStruct, double x,
  double y, double W, double H, int numberOfLines, double THETA, int
  gdsLayer, double shapeReso)

// resoPatternLSA
public static ArrayList<GArea> createResoPatternLSA(double x, double y,
  double startW, double endW, double delta, double H, int
  numberOfLines, double space, double THETA, int gdsLayer, double
  shapeReso)

// resoPatternLSA
public static void createResoPatternLSA(Struct currentStruct, double x,
  double y, double startW, double endW, double delta, double H, int
  numberOfLines, double space, double THETA, int gdsLayer, double
  shapeReso)

// spiralArch
public static ArrayList<GArea> createSpiralArch(double x, double y,
  double width, int turns, double separation, double inc, double
  THETA, int gdsLayer)

// spiralArch
public static void createSpiralArch(Struct currentStruct, double x,
  double y, double width, int turns, double separation, double inc,
  double THETA, int gdsLayer)

// spiralFermat
public static ArrayList<GArea> createSpiralFermat(double x, double y,
  double width, int turns, double a, double inc, double THETA, int
gdsLayer)

// spiralFermat
public static void createSpiralFermat(Struct currentStruct, double x,
  double y, double width, int turns, double a, double inc, double
  THETA, int gdsLayer)
public static ArrayList<GArea> createSpiralLog(double x, double y,
    double width, int turns, double a, double b, double inc, double
    THETA, int gdsLayer)

public static void createSpiralLog(Struct currentStruct, double x,
    double y, double width, int turns, double a, double b, double inc,
    double THETA, int gdsLayer)

public static ArrayList<GArea> createSpiralRect(double x, double y,
    double width, double len, double pitch, int numTurns, double THETA,
    int gdsLayer)

public static void createSpiralRect(Struct currentStruct, double x,
    double y, double width, double len, double pitch, int numTurns,
    double THETA, int gdsLayer)
4.2.7 Alignment and Reticle Element Methods

// alignFFFB1
public static ArrayList<GArea> createAlignFFFB1(double x, double y, int layerCross, int layerSquare, double THETA)

// alignFFFB1
public static void createAlignFFFB1(Struct currentStruct, double x, double y, int layerCross, int layerSquare, double THETA)

// alignFFFB2
public static ArrayList<GArea> createAlignFFFB2(double x, double y, int layerCross, int layerSquare, double THETA)

// alignFFFB2
public static void createAlignFFFB2(Struct currentStruct, double x, double y, int layerCross, int layerSquare, double THETA)

// align3Level
public static ArrayList<GArea> createAlign3Level(double x, double y, int layerA, int layerB, int layerC, double THETA)

// align3Level
public static void createAlign3Level(Struct currentStruct, double x, double y, int layerA, int layerB, int layerC, double THETA)

// alignVern
public static ArrayList<GArea> createAlignVern(double x, double y, int layer1, int layer2, double vernReso, boolean L1INV, boolean L2INV, double invL, double invW, double THETA)

// alignVern
public static void createAlignVern(Struct currentStruct, double x, double y, int layer1, int layer2, double vernReso, boolean L1INV, boolean L2INV, double invL, double invW, double THETA)

// alignVernLb1
public static ArrayList<GArea> createAlignVernLb1(double x, double y, int layer1, int layer2, double vernReso, double THETA)

// alignVernLb1
public static void createAlignVernLb1(Struct currentStruct, double x, double y, int layer1, int layer2, double vernReso, double THETA)
// alignVernLb2
public static ArrayList<GArea> createAlignVernLb2(double x, double y,
        int layer1, int layer2, double vernReso, double THETA)

// alignVernLb2
public static void createAlignVernLb2(Struct currentStruct, double x,
        double y, int layer1, int layer2, double vernReso, double THETA)

// alignCustC1
public static GArea createCustomCrossC1(double x, double y, double L1,
        double W1, double L2, boolean INVERSE, double IL, double IW, int
        TYPE, double THETA, int gdsLayer)

// alignCustC2
public static GArea createCustomCrossC2(double x, double y, double L1,
        double W1, double PL, double PW, boolean INVERSE, double IL, double
        IW, TYPE, double THETA, int gdsLayer)

// alignCustC3
public static GArea createCustomCrossC3(double x, double y, double L1,
        double W1, double PL, double PW, double LX, boolean INVERSE, double
        IL, double IW, TYPE, double THETA, int gdsLayer)

// alignCustC4
public static GArea createCustomCrossC4(double x, double y, double L1,
        double W1, double L2, double d, double BL, double BW, boolean
        INVERSE, double IL, double IW, TYPE, double THETA, int gdsLayer)

// cnstASML
public static ArrayList<GArea> createCnstASML(String label, String
        barcode, int gdsLayer) throws FileNotFoundException

// cnstASML
public static void createCnstASML(Struct currentStruct, String label,
        String barcode, boolean MIRROR, int gdsLayer) throws
        FileNotFoundException

// cnstContact5
public static ArrayList<GArea> createCnstContact5(String label, int
        gdsLayer) throws FileNotFoundException

// cnstContact5
public static void createCnstContact5(Struct currentStruct, String
        label, boolean MIRROR, int gdsLayer) throws FileNotFoundException

// cnstContact7
public static ArrayList<GArea> createCnstContact7(String label, int gdsLayer) throws FileNotFoundException

// cnstContact7
public static void createCnstContact7(Struct currentStruct, String label, boolean MIRROR, int gdsLayer) throws FileNotFoundException

// alignCustVern
public static ArrayList<GArea> createCustomVerniers(double x, double y, int layer1, int layer2, double L, double W, double pitch, double vernReso, double spacing, boolean INVERSE1, boolean INVERSE2, double IL, double IW, double THETA, double shapeReso)

// alignCustVern
public static void createCustomVerniers(Struct currentStruct, double x, double y, int layer1, int layer2, double L, double W, double pitch, double vernReso, double spacing, boolean INVERSE1, boolean INVERSE2, double IL, double IW, double THETA, int gdsLayer, double shapeReso)

// verniers − custom verniers with text labels
public static ArrayList<GArea> createVerniers(double x, double y, int layer1, int layer2, double vernierReso, int ticks, String label1, String label2, double fontSize, double fontReso, double lineWidth, double lineLength, double pitch, double THETA)

// verniers − custom verniers with text labels
public static void createVerniers(Struct currentStruct, double x, double y, int layer1, int layer2, double vernierReso, int ticks, String label1, String label2, double fontSize, double fontReso, double lineWidth, double lineLength, double pitch, double THETA)

// arrowHead
public static GArea createArrowHead(double x, double y, double arrowW, double arrowL, double width, double THETA, int gdsLayer)

// arrow
public static GArea createArrow(double x, double y, double arrowW, double arrowL, double width, double length, double THETA, int gdsLayer)

// arrowArray
public static GArea createArrowLinearArray(double x, double y, double arrowW, double arrowL, double width, double length, int numberOfArrows, double THETA, int gdsLayer)
4.2.8 Nanophotonics Library Methods

// waveguide
public static GArea createWaveGuide(double x1, double y1, double x2, double y2, double w, double THETA, int ENDCAPLEFT, int ENDCAPRIGHT, int gdsLayer)

// waveguideSlot
public static GArea createWaveGuideSlot(double x1, double y1, double x2, double y2, double w, double slotWidth, double THETA, int ENDCAPLEFT, int ENDCAPRIGHT, int gdsLayer)

// waveguideInv
public static GArea createWaveGuideInv(double x1, double y1, double x2, double y2, double w, double invWidth, double THETA, int ENDCAPLEFT, int ENDCAPRIGHT, int gdsLayer)

// waveguideInvSlot
public static GArea createWaveGuideInvSlot(double x1, double y1, double x2, double y2, double w, double slotWidth, double invWidth, double THETA, int ENDCAPLEFT, int ENDCAPRIGHT, int gdsLayer)

// wgExpander
public static GArea createWaveGuideExpander(double x, double y, double width, double length, double deltaLength, double a, double b, double angle, double w0, double lambda, double baseHeight, double THETA, int gdsLayer)

// linearTaper
public static GArea createLinearTaper(double x1, double y1, double x2, double y2, double w1, double w2, double THETA, int gdsLayer)

// linearTaperSlot
public static ArrayList<GArea> createLinearTaperSlot(double x1, double y1, double x2, double y2, double w1, double w2, double slotWidth1, double slotWidth2, double THETA, int gdsLayer)

// linearTaperInvSlot
public static void createLinearTaperInvSlot(ArrayList<GArea> list, Struct currentStruct, double x1, double y1, double x2, double y2, double w1, double w2, double slotWidth1, double slotWidth2, double THETA, int gdsLayer)
```java
// linearTaperInvSlot
public static void createLinearTaperInvSlot(Struct currentStruct,
   double x1, double y1, double x2, double y2, double w1, double w2,
   double gap, double slotWidth1, double slotWidth2, double THETA, int
   gdsLayer)

// exponentialTaper
public static GArea createExponentialTaper(double x1, double y1, double
   L, double W1, double W2, int numPoints, double THETA,int gdsLayer,
   double shapeReso)

// exponentialTaperInv
public static ArrayList<GArea> createExpTaperInv(double x1, double y1,
   double length, double wStart, double wEnd, int nPoints, double
   widthAround, double THETA, int gdsLayer, double shapeReso)

// exponentialTaperInv
public static ArrayList<GArea> createExpTaperInvSlot(double x1, double
   y1, double length, double wStart, double wEnd, int nPoints, double
   inverseWidth, double slotW1, double slotW2, double THETA, int
   gdsLayer, double shapeReso)

// exponentialTaperInvSlot
public static ArrayList<GArea> createExpTaperInvSlot(Struct currentStruct, double
   x1, double y1, double length, double wStart, double wEnd, int nPoints, double
   inverseWidth, double slotW1, double slotW2, double THETA, int
   gdsLayer, double shapeReso)

// customTaper
public static GArea createCustomTaper(ArrayList<Double> al, double Tx,
   double Ty, double THETA, int gdsLayer)

// directionalCoupler1
public static GArea createDirectionalCoupler1(double x, double y,
   double l1, double w1, double we1, double l2, double l3, double w2,
   double we2, double g, double r, int numSides, double THETA, int
   gdsLayer)

// directionalCoupler2
public static GArea createDirectionalCoupler2(double x, double y,
   double l1, double w1, double we1, double l2, double l3, double w2,
   double we2, double g, double r, int numSides, double THETA, int
   gdsLayer)
```
// directionalCoupler3
public static GArea createDirectionalCoupler3(double x, double y,
    double w, double wE, double g, double L1, double L2, double r,
    int numSides, double THETA, int gdsLayer)

// directionalCoupler4
public static GArea createDirectionalCoupler4(double x, double y,
    double w, double wE, double g, double L1, double LB, double HB,
    double THETA, int gdsLayer)

// sBendTaper
public static GArea createSbendTaper(double x, double y, double L,
    double H, double wStart, double wEnd, double THETA, int gdsLayer,
    double shapeReso)

// sBendFunnel
public static GArea createSbendFunnel(double x, double y, double L,
    double H, double W, double THETA, int gdsLayer, double shapeReso)

// sBend
public static GArea createSbend(double x1, double y1, double x2, double
    y2, double width, double THETA, int gdsLayer, double shapeReso)

// sBendLH
public static GArea createSbendLH(double x1, double y1, double length,
    double height, double width, double THETA, int gdsLayer, double
    shapeReso)

// sBendInv
public static GArea createSbendInv(double x1, double y1, double length,
    double height, double width, double sleeve, double THETA, int
    gdsLayer, double shapeReso)

// sBendInvSlot
public static GArea createSbendInvSlot(double x, double y, double
    length, double height, double slotWidth, double gap, double sleeve,
    double THETA, int gdsLayer, double shapeReso)

// yBend
public static GArea createYbend(double x1, double y1, double x2, double
    y2, double x3, double y3, double width, double THETA, int gdsLayer,
    double shapeReso)

// yBendLH
public static GArea createYbendLH(double x1, double y1, double length2,
    double height2, double length3, double height3, double width,
    double THETA, int gdsLayer, double shapeReso)
// yBendInv
public static GArea createYbendInv(double x1, double y1, double length2,
  double height2, double length3, double height3, double width,
  double sleeve, double THETA, int gdsLayer, double shapeReso)

// yBendInvSlot
public static GArea createYbendInvSlot(double x1, double y1, double
  length2, double height2, double length3, double height3, double
  slotWidth, double gap, double sleeve, double THETA, int gdsLayer,
  double shapeReso)

// yBend90
public static GArea createYbend90(double x, double y, double
  r1, double r2, double w, int numSides, double THETA, int gdsLayer)

// yBendInv90
public static GArea createYBendInv90(double x, double y, double
  r1, double r2, double w, double we, int numSides, double THETA,
  int gdsLayer)

// yBendInvSlot90
public static GArea createYBendInvSlot90(double x, double y, double
  r1, double r2, double ws, double g, double we, int numSides, double
  THETA, int gdsLayer)

// 90degreeBend
public static GArea create90degreeBend(double x1, double y1, double
  x2, double y2, double width, double THETA, int gdsLayer, double
  shapeReso)

// 90degreeBendLH
public static GArea create90degreeBendLH(double x1, double y1, double
  length, double height, double width, double THETA, int gdsLayer,
  double shapeReso)

// 90degreeBendInv
public static GArea create90degreeInv(double x1, double y1, double
  length, double height, double width, double sleeve, double THETA,
  int gdsLayer, double shapeReso)

// 90degreeBendInvSlot
public static GArea create90degreeInvSlot(double x1, double y1, double
  length, double height, double slotWidth, double gap, double sleeve,
  double THETA, int gdsLayer, double shapeReso)

// 180degreeBend
public static GArea create180bend(double x, double y, double L1, double
public static GArea create180bendInv(double x, double y, double L1, double L2, double D, double W, double We, int numSides, double THETA, int gdsLayer)

public static GArea create180bendInvSlot(double x, double y, double L1, double L2, double D, double W, double gap, double We, int numSides, double THETA, int gdsLayer)

public static GArea createRaceTrack(double x, double y, double length, double width, double radiusInner, double THETA, int numSides, int gdsLayer)

public static ArrayList<AreaList> createSpiralDelayLineArchV2(Struct currentStruct, double x, double y, double width, int turns, double pitch, int stepsPerTurn, int skippedTurns, double length, double THETA, int ENDCAP, int gdsLayer)

public static ArrayList<AreaList> createSpiralDelayLineArchInv(Struct currentStruct, double x, double y, double width, double sleeveWidth, int turns, double separation, double deltaR, double length, double THETA, int ENDCAP, int gdsLayer)
// spiralDelayLineArchInv
public static void createSpiralDelayLineArchInv(Struct currentStruct,
        double x, double y, double width, double sleeveWidth, int turns,
        double separation, double deltaR, double length, double THETA, int ENDCAP, int gdsLayer)

// spiralDelayLineFermatInv
public static ArrayList<GArea> createSpiralDelayLineFermatInv(double x,
        double y, double width, double sleeveWidth, int turns, double a,
        double deltaR, double length, double THETA, int ENDCAP, int gdsLayer)

// spiralDelayLineArchV2Inv
public static void createSpiralDelayLineArchV2Inv(Struct currentStruct,
        double x, double y, double width, double sleeveWidth, int turns,
        double pitch, int stepsPerTurn, int skippedTurns, double length,
        double THETA, int ENDCAP, int gdsLayer)

// grating
public static void createGrating(Struct currentStruct, String name,
        double x, double y, double width, double length, double pitch, int
        numberOfLines, int gdsLayer)

// gratingCoupler
public static void createGratingCoupler(Struct currentStruct, double x,
        double y, double r, double W, double pitch, double angleStart,
        double angleEnd, int numSides, int numElements, int gdsLayer)

// apodizedGrating
public static GArea createApodizedGrating(Struct currentStruct, double
        x, double y, double gratingLength, double increment, double height,
        double dutyCycleCutoff, double p1, double p2, double p3, double
        p4, double d1, double d2, double d3, double d4, double d5, double
        THETA, int gdsLayer)

// gratingCWG
public static ArrayList<GArea> createGratingCWG(double x, double y,
        double wgWidth, double Length, double Height, double sleeve, double
        lambda0, double nEff, double nCladding, double thetaC, double R1,
        double gratingPeriod, double ratio, int numberOfElements, int
        numSides, int layerWaveGuide, int layerGrating, boolean ENDCAPS,
        double THETA, int gdsLayer, double shapeReso)
// gratingCWG
public static void createGratingCWG(Struct currentStruct, double x,
double y, double wgWidth, double Length, double Height, double
sleeve, double lambda0, double nEff, double nCladding, double
thetaC, double R1, double gratingPeriod, double ratio, int
numberOfElements, int numSides, int layerWaveGuide, int
layerGrating, boolean ENDCAPS, double THETA, int gdsLayer, double
shapeReso)

// gratingCWGinv
public static ArrayList<GArea> createGratingCWGinv(double x, double y,
double wgWidth, double Length, double Height, double sleeve, double
lambda0, double nEff, double nCladding, double thetaC, double R1,
double gratingPeriod, double ratio, int numberOfElements, int
numSides, int layerWaveGuide, int layerGrating, boolean ENDCAPS,
double THETA, int gdsLayer, double shapeReso)

// phC
public static void createPhC(Struct currentStuct, String
uniqueStructName, double x, double y, double radius, double pitchX,
int elementsX, int elementsY, int numberOfSides, double
arrayPitchDelta, int gdsLayer)

// phCV
public static void createPhCV(Struct currentStuct, String
uniqueStructName, double x, double y, double radius, double pitchX,
int elementsX, int elementsY, int numberOfSides, double
arrayPitchDelta, int gdsLayer, double shapeReso)

// hex
public static void createHex(Struct currentStuct, String
uniqueStructName, double x, double y, double radius, double pitchX,
int elementsX, int elementsY, int numberOfSides, int gdsLayer)

// hexV
public static void createHexV(Struct currentStuct, String
uniqueStructName, double x, double y, double radius, double pitchX,
int elementsX, int elementsY, int numberOfSides, int gdsLayer,
double shapeReso)
// discInfinite
public static void createDiscInfinite(Struct currentStuct, double x, 
double y, double radius, int numberOfSidesDiscRing, double gap, 
double waveguideLength, double waveguideWidth, boolean ENDCAP, int 
gdsLayer)

// ringInfinite
public static void createRingInfinite(Struct currentStuct, double x, 
double y, double radius, double ringWidth, int 
numberOfSidesDiscRing, double gap, double waveguideLength, double 
waveguideWidth, boolean ENDCAP, int gdsLayer)

// discInfiniteDS
public static void createDiscInfiniteDS(Struct currentStuct, double x, 
double y, double radius, int numberOfSidesDiscRing, double gap, 
double waveguideLength, double waveguideWidth, double gap2, double 
length2, double width2, boolean ENDCAP, int gdsLayer)

// ringInfiniteDS
public static void createRingInfiniteDS(Struct currentStuct, double x, 
double y, double radius, double ringWidth, int 
numberOfSidesDiscRing, double gap, double waveguideLength, double 
waveguideWidth, double gap2, double length2, double width2, boolean 
ENDCAP, int gdsLayer)

// discInfiniteInv
public static void createDiscInfiniteInv(Struct currentStuct, double x, 
double y, double radius, int numberOfSidesDiscRing, double gap, 
double waveguideLength, double waveguideWidth, double sleeveWidth, 
boolean ENDCAP, int gdsLayer)

// ringInfiniteInv
public static void createRingInfiniteInv(Struct currentStuct, double x, 
double y, double radius, double ringWidth, int 
numberOfSidesDiscRing, double gap, double waveguideLength, double 
waveguideWidth, double sleeveWidth, boolean ENDCAP, int gdsLayer)

// discInfiniteDSInv
public static void createDiscInfiniteDSInv(Struct currentStuct, double x, 
double y, double radius, int numberOfSidesDiscRing, double gap, 
double waveguideLength, double waveguideWidth, double sleeveWidth, 
double gap2, double length2, double width2, double sleeve2, boolean 
ENDCAP, int gdsLayer)

// ringInfiniteDSInv
public static void createRingInfiniteDSInv(Struct currentStuct, double x, 
double y, double radius, double ringWidth, int 
numberOfSidesDiscRing, double gap, double waveguideLength, double 
waveguideWidth, double sleeveWidth, double gap2, double length2, 
double width2, double sleeve2, boolean ENDCAP, int gdsLayer)

NIST • CNST Nanolithography Toolbox v2016.09.01 • http://www.nist.gov/cnst/  page 424 of 488
// discInfiniteInvPos
public static void createDiscInfiniteInvPos(Struct currentStuct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideLength, double waveguideWidth, double sleeveWidth, boolean ENDCAP, int gdsLayer)

// ringInfiniteInvPos
public static void createRingInfiniteInvPos(Struct currentStuct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideLength, double waveguideWidth, double sleeveWidth, boolean ENDCAP, int gdsLayer)

// discInfiniteInvPosDS
public static void createDiscInfiniteInvPosDS(Struct currentStuct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideLength, double waveguideWidth, double sleeveWidth, double gap2, double length2, double width2, double sleeve2, boolean ENDCAP, int gdsLayer)

// ringInfiniteInvPosDS
public static void createRingInfiniteInvPosDS(Struct currentStuct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideLength, double waveguideWidth, double sleeveWidth, double gap2, double length2, double width2, double sleeve2, boolean ENDCAP, int gdsLayer)

// discSymmetric
public static void createDiscSymmetric(Struct currentStuct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymmetric
public static void createRingSymmetric(Struct currentStuct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymDS
public static void createDiscSymDS(Struct currentStuct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)
// ringSymDS
public static void createRingSymDS(Struct currentStruct, double x,
    double y, double radius, double ringWidth, int
    numberOfSidesDiscRing, double gap, double waveguideAngle, int
    numberOfSidesCouplingRegion, double waveguideWidth, double
    waveguideLength, double waveguideHeight, double gap2, double width2
    , double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymPul
public static void createDiscSymPul(Struct currentStruct, double x,
    double y, double radius, int numberOfSidesDiscRing, double gap,
    double waveguideAngle, int numberOfSidesCouplingRegion, double
    waveguideWidth, double waveguideLength, double waveguideHeight,
    double gap2, double angle2, double width2, double length2, double
    height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymPul
public static void createRingSymPul(Struct currentStruct, double x,
    double y, double radius, double ringWidth, int
    numberOfSidesDiscRing, double gap, double waveguideAngle, int
    numberOfSidesCouplingRegion, double waveguideWidth, double
    waveguideLength, double waveguideHeight, double gap2, double angle2
    , double width2, double length2, double height2, boolean ENDCAP,
    int gdsLayer, double shapeReso)

// discSymmetricLC
public static void createDiscSymmetricLC(Struct currentStruct, double x
    , double y, double radius, double ringWidth, int
    numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion,
    double waveguideWidth, double waveguideLength, double waveguideHeight,
    boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymmetricLC
public static void createRingSymmetricLC(Struct currentStruct, double x
    , double y, double radius, double ringWidth, int
    numberOfSidesDiscRing, double gap, double Lc, int
    numberOfSidesCouplingRegion, double waveguideWidth, double
    waveguideLength, double waveguideHeight, boolean ENDCAP, int
    gdsLayer, double shapeReso)

// discSymLCDS
public static void createDiscSymLCDS(Struct currentStruct, double x,
    double y, double radius, int numberOfSidesDiscRing, double gap,
    double Lc, int numberOfSidesCouplingRegion, double waveguideWidth,
    double waveguideLength, double waveguideHeight, boolean ENDCAP, int
    gdsLayer, double shapeReso)
public static void createRingSymLCDS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscSymLCPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createRingSymLCPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscSymmetricA(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createRingSymmetricA(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createDiscSymADS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer)

public static void createRingSymADS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer)
// discSymAPul
public static void createDiscSymAPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double angle2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// ringSymAPul
public static void createRingSymAPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double angle2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// discSymmetricLCA
public static void createDiscSymmetricLCA(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// ringSymmetricLCA
public static void createRingSymmetricLCA(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// discSymLCADS
public static void createDiscSymLCADS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// ringSymLCADS
public static void createRingSymLCADS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// discSymLCAPul
public static void createDiscSymLCAPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth,
double waveguideLength, double gap2, double Lc2, double width2,
double length2, boolean ENDCAP, int gdsLayer)

// ringSymLCAPul
public static void createRingSymLCAPul(Struct currentStruct, double x,
double y, double radius, double ringWidth, int
numberOfSidesDiscRing, double gap, double Lc1, int
numberOfSidesCouplingRegion, double waveguideWidth, double
waveguideLength, double gap2, double Lc2, double width2, double
length2, boolean ENDCAP, int gdsLayer)

// discSymmetricInv
public static void createDiscSymmetricInv(Struct currentStruct, double x,
double y, double radius, int numberOfSidesDiscRing, double gap,
double waveguideAngle, int numberOfSidesCouplingRegion, double
waveguideWidth, double sleeveWidth, double waveguideLength, double
waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymmetricInv
public static void createRingSymmetricInv(Struct currentStruct, double x,
double y, double radius, double ringWidth, int
numberOfSidesDiscRing, double gap, double waveguideAngle, int
numberOfSidesCouplingRegion, double waveguideWidth, double
sleeveWidth, double waveguideLength, double waveguideHeight, double
gap2, double width2, double sleeve2, double length2, boolean
ENDCAP, int gdsLayer, double shapeReso)

// discSymInvDS
public static void createDiscSymInvDS(Struct currentStruct, double x,
double y, double radius, int numberOfSidesDiscRing, double gap,
double waveguideAngle, int numberOfSidesCouplingRegion, double
waveguideWidth, double sleeveWidth, double waveguideLength, double
waveguideHeight, double gap2, double width2, double sleeve2, double
length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymInvDS
public static void createRingSymInvDS(Struct currentStruct, double x,
double y, double radius, double ringWidth, int
numberOfSidesDiscRing, double gap, double waveguideAngle, int
numberOfSidesCouplingRegion, double waveguideWidth, double
sleeveWidth, double waveguideLength, double waveguideHeight, double
gap2, double width2, double sleeve2, double length2, boolean
ENDCAP, int gdsLayer, double shapeReso)

// discSymInvPul
public static void createDiscSymInvPul(Struct currentStruct, double x,
double y, double radius, int numberOfSidesDiscRing, double gap,
double waveguideAngle, int numberOfSidesCouplingRegion, double
waveguideWidth, double sleeveWidth, double waveguideLength, double
waveguideHeight, double gap2, double angle2, double width2, double
sleeve2, double length2, double height2, boolean ENDCAP, int
gdsLayer, double shapeReso)
// ringSymInvPul
public static void createRingSymInvPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double angle2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymmetricInvLC
public static void createDiscSymmetricInvLC(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymmetricInvLC
public static void createRingSymmetricInvLC(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymInvLCDS
public static void createDiscSymInvLCDS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymInvLCDS
public static void createRingSymInvLCDS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymInvLCPul
public static void createDiscSymInvLCPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)
// ringSymInvLCPul
public static void createRingSymInvLCPul(Struct currentStruct, double x,
    double y, double radius, double ringWidth, int
    numberOfSidesDiscRing, double gap, double Lc1, int
    numberOfSidesCouplingRegion, double waveguideWidth, double
    sleeveWidth, double waveguideLength, double waveguideHeight, double
    gap2, double Lc2, double width2, double sleeve2, double length2,
    double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymmetricInvA
public static void createDiscSymmetricInvA(Struct currentStruct, double
    x, double y, double radius, int numberOfSidesDiscRing, double gap,
    double waveguideAngle, int numberOfSidesCouplingRegion, double
    waveguideWidth, double sleeveWidth, double waveguideLength, boolean
    ENDCAP, int gdsLayer)

// ringSymmetricInvA
public static void createRingSymmetricInvA(Struct currentStruct, double
    x, double y, double radius, double ringWidth, int
    numberOfSidesDiscRing, double gap, double waveguideAngle, int
    numberOfSidesCouplingRegion, double waveguideWidth, double
    sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// discSymInvADS
public static void createDiscSymInvADS(Struct currentStruct, double x,
    double y, double radius, int numberOfSidesDiscRing, double gap,
    double waveguideAngle, int numberOfSidesCouplingRegion, double
    waveguideWidth, double sleeveWidth, double waveguideLength, double
    gap2, double width2, double sleeve2, double length2, boolean ENDCAP,
    int gdsLayer)

// ringSymInvADS
public static void createRingSymInvADS(Struct currentStruct, double x,
    double y, double radius, double ringWidth, int
    numberOfSidesDiscRing, double gap, double waveguideAngle, int
    numberOfSidesCouplingRegion, double waveguideWidth, double
    sleeveWidth, double waveguideLength, double gap2, double width2,
    double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// discSymInvAPul
public static void createDiscSymInvAPul(Struct currentStruct, double x,
    double y, double radius, int numberOfSidesDiscRing, double gap,
    double waveguideAngle, int numberOfSidesCouplingRegion, double
    waveguideWidth, double sleeveWidth, double waveguideLength, double
    gap2, double angle2, double width2, double sleeve2, double length2,
    boolean ENDCAP, int gdsLayer)

// ringSymInvAPul
public static void createRingSymInvAPul(Struct currentStruct, double x,
    double y, double radius, double ringWidth, int
public static void createDiscSymmetricInvLCA (Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createRingSymmetricInvLCA (Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createDiscSymInvLCADS (Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createRingSymInvLCADS (Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createDiscSymInvLCAPul (Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createRingSymInvLCAPul (Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)
// discSymmetricInvPos
public static void createDiscSymmetricInvPos(Struct currentStruct,
double x, double y, double radius, double ringDiscSleeveWidth, int
numberOfSidesDiscRing, double gap, double waveguideAngle, int
numberOfSidesCouplingRegion, double waveguideWidth, double
sleeveWidth, double waveguideLength, double waveguideHeight,
boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymmetricInvPos
public static void createRingSymmetricInvPos(Struct currentStruct,
double x, double y, double radius, double ringWidth, double
ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double
waveguideAngle, int numberOfSidesCouplingRegion, double
waveguideWidth, double sleeveWidth, double waveguideLength, double
waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymInvPosDS
public static void createDiscSymInvPosDS(Struct currentStruct, double x
, double y, double radius, double ringDiscSleeveWidth, int
numberOfSidesDiscRing, double gap, double waveguideAngle, int
numberOfSidesCouplingRegion, double waveguideWidth, double
sleeveWidth, double waveguideLength, double waveguideHeight, double
gap2, double width2, double sleeve2, double length2, boolean
ENDCAP, int gdsLayer, double shapeReso)

// ringSymInvPosDS
public static void createRingSymInvPosDS(Struct currentStruct, double x
, double y, double radius, double ringWidth, double
ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double
waveguideAngle, int numberOfSidesCouplingRegion, double
waveguideWidth, double sleeveWidth, double waveguideLength, double
waveguideHeight, double gap2, double width2, double sleeve2, double
length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymInvPosPul
public static void createDiscSymInvPosPul(Struct currentStruct, double x,
double y, double radius, double ringDiscSleeveWidth, int
numberOfSidesDiscRing, double gap, double waveguideAngle, int
numberOfSidesCouplingRegion, double waveguideWidth, double
gap2, double angle2, double width2, double sleeve2, double length2,
double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymInvPosPul
public static void createRingSymInvPosPul(Struct currentStruct, double x,
double y, double radius, double ringDiscSleeveWidth, int
numberOfSidesDiscRing, double gap, double waveguideAngle, int
numberOfSidesCouplingRegion, double waveguideWidth, double
waveguideHeight, double waveguideLength, double
waveguideWidth, double sleeveWidth, double waveguideLength, double
waveguideHeight, double gap2, double angle2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymmetricInvPosLC
public static void createDiscSymmetricInversePosLC(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymmetricInvPosLC
public static void createRingSymmetricInversePosLC(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymInvPosLCDS
public static void createDiscSymInvPosLCDS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymInvPosLCDS
public static void createRingSymInvPosLCDS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymInvPosLCPul
public static void createDiscSymInvPosLCPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringSymInvPosLCPul
public static void createRingSymInvPosLCPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)
Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discSymmetricInvPosA
public static void createDiscSymmetricInvPosA(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// ringSymmetricInvPosA
public static void createRingSymmetricInvPosA(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// discSymInvPosADS
public static void createDiscSymInvPosADS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// ringSymInvPosADS
public static void createRingSymInvPosADS(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// discSymInvPosAPul
public static void createDiscSymInvPosAPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double angle2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// ringSymInvPosAPul
public static void createRingSymInvPosAPul(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)
public static void createDiscSymmetricInvPosLCA(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberofSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createRingSymmetricInvPosLCA(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberofSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createDiscSymInvPosLCADS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberofSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createRingSymInvPosLCADS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberofSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createDiscSymInvPosLCAPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberofSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createRingSymInvPosLCAPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberofSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)
// discPulley
public static void createDiscPulley(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPulley
public static void createRingPulley(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPulDS
public static void createDiscPulDS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPulDS
public static void createRingPulDS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPulPul
public static void createDiscPulPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double angle2, double width2, double length2, double heigth2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPulPul
public static void createRingPulPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double angle2, double width2, double length2, double heigth2, boolean ENDCAP, int gdsLayer, double shapeReso)
public static void createDiscPulleyLC(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createRingPulleyLC(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPulLCDS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createRingPulLCDS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPulLCPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createRingPulLCPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPulleyA(Struct currentStruct, double x,
double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// ringPulleyA
public static void createRingPulleyA(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// discPulADS
public static void createDiscPulADS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// ringPulADS
public static void createRingPulADS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// discPulAPul
public static void createDiscPulAPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// ringPulAPul
public static void createRingPulAPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, double gap2, double angle2, double width2, double length2, boolean ENDCAP, int gdsLayer)

// discPulleyLCA
public static void createDiscPulleyLCA(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// ringPulleyLCA
public static void createRingPulleyLCA(Struct currentStruct, double x,
double y, double radius, double ringWidth, int
numberOfSidesDiscRing, double gap, double Lc, int
numberOfSidesCouplingRegion, double waveguideWidth, double
waveguideLength, boolean ENDCAP, int gdsLayer)

// discPulLCADS
public static void createDiscPulLCADS(Struct currentStruct, double x,
double y, double radius, int numberOfSidesDiscRing, double gap,
double Lc, int numberOfSidesCouplingRegion, double waveguideWidth,
double waveguideLength, double gap2, double width2, double length2,
boolean ENDCAP, int gdsLayer)

// ringPulLCADS
public static void createRingPulLCADS(Struct currentStruct, double x,
double y, double radius, double ringWidth, int
numberOfSidesDiscRing, double gap, double Lc, int
numberOfSidesCouplingRegion, double waveguideWidth, double
waveguideLength, double gap2, double width2, double length2,
boolean ENDCAP, int gdsLayer)

// discPulLCAPul
public static void createDiscPulLCAPul(Struct currentStruct, double x,
double y, double radius, int numberOfSidesDiscRing, double gap,
double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth,
double waveguideLength, double gap2, double Lc2, double width2,
double length2, boolean ENDCAP, int gdsLayer)

// ringPulLCAPul
public static void createRingPulLCAPul(Struct currentStruct, double x,
double y, double radius, double ringWidth, int
numberOfSidesDiscRing, double gap, double Lc1, int
numberOfSidesCouplingRegion, double waveguideWidth, double
waveguideLength, double gap2, double Lc2, double width2, double
length2, boolean ENDCAP, int gdsLayer)

// discPulleyInv
public static void createDiscPulleyInverse(Struct currentStruct, double x,
double y, double radius, int numberOfSidesDiscRing, double gap,
double waveguideAngle, int numberOfSidesCouplingRegion, double
waveguideWidth, double sleeveWidth, double waveguideLength, double
waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPulleyInv
public static void createRingPulleyInverse(Struct currentStruct, double x,
double y, double radius, double ringWidth, int
numberOfSidesDiscRing, double gap, double waveguideAngle, int
numberOfSidesCouplingRegion, double waveguideWidth, double
sleeveWidth, double waveguideLength, double waveguideHeight,
boolean ENDCAP, int gdsLayer, double shapeReso)
// discPullInvDS
public static void createDiscPullInvDS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPullInvDS
public static void createRingPullInvDS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPullInvPul
public static void createDiscPullInvPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double angle2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPullInvPul
public static void createRingPullInvPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double angle2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPulleyInvLC
public static void createDiscPulleyInvLC(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPulleyInvLC
public static void createDiscPulleyInverseLC(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPullInvLCDS
public static void createDiscPullInvLCDS(Struct currentStruct, double x,
public static void createRingPulInvLCDS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPulInvLCPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createRingPulInvLCPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPulleyInvA(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createRingPulleyInvA(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createDiscPulInvADS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer, double shapeReso)
public static void createRingPulInvADS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createDiscPulInvAPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createRingPulInvAPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createDiscPulleyInvLCA(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createRingPulleyInvLCA(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

public static void createDiscPulInvLCADS(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)
// ringPulInvLCADS
public static void createRingPulInvLCADS(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// discPulInvLCAPul
public static void createDiscPulInvLCAPul(Struct currentStruct, double x, double y, double radius, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// ringPulInvLCAPul
public static void createRingPulInvLCAPul(Struct currentStruct, double x, double y, double radius, double ringWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// discPulleyInvPos
public static void createDiscPulleyInvPos(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPulleyInvPos
public static void createRingPulleyInvPos(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPulInvPDS
public static void createDiscPulInvPDS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)
public static void createRingPullInvPDS(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPullInvPPLPul(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double angle2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createRingPullInvPPLPul(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double angle2, double width2, double sleeve2, double length2, double height2, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPulleyInvPosLC(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double LC, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createRingPulleyInvPosLC(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double LC, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)

public static void createDiscPullInvPCLCDS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double LC, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, boolean ENDCAP, int gdsLayer, double shapeReso)
// ringPullInvPLCDS
public static void createRingPullInvPLCDS(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPullInvPLCPul
public static void createDiscPullInvPLCPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// ringPullInvPLCPul
public static void createRingPullInvPLCPul(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double waveguideHeight, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer, double shapeReso)

// discPulleyInvPosA
public static void createDiscPulleyInvPosA(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// ringPulleyInvPosA
public static void createRingPulleyInvPosA(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)

// discPullInvPADS
public static void createDiscPullInvPADS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, boolean ENDCAP, int gdsLayer)
sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// ringPullInvPADS
public static void createRingPullInvPADS(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// discPullInvPAPul
public static void createDiscPullInvPAPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double waveguideAngle, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double angle2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// ringPulleyInvPosLCA
public static void createRingPulleyInvPosLCA(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, boolean ENDCAP, int gdsLayer)

// discPulleyInvPosLCA
public static void createDiscPulleyInvPosLCA(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, boolean ENDCAP, int gdsLayer)

// discPullInvPLCADS
public static void createDiscPullInvPLCADS(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)
public static void createRingPulInvPLCADS(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createDiscPulInvPLCAPul(Struct currentStruct, double x, double y, double radius, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

public static void createRingPulInvPLCAPul(Struct currentStruct, double x, double y, double radius, double ringWidth, double ringDiscSleeveWidth, int numberOfSidesDiscRing, double gap, double Lc1, int numberOfSidesCouplingRegion, double waveguideWidth, double sleeveWidth, double waveguideLength, double gap2, double Lc2, double width2, double sleeve2, double length2, boolean ENDCAP, int gdsLayer)

// waveGuideInvPhC – al is a Double vector array containing rx1 ry1 N1 .... rxn ryn Nn values
public static ArrayList<GArea> createWaveGuideInvPhC(ArrayList<Double> al, double x, double y, double a, double W, int gdsLayer)

// waveGuideInvPhC – al is a Double vector array containing rx1 ry1 N1 .... rxn ryn Nn values
public static void createWaveGuideInvPhC(Struct currentStruct, ArrayList<Double> al, double x, double y, double a, double W, int gdsLayer)

// waveGuideInvPhCvary – al is a Double vector array containing rx1 ry1 N1 .... rxn ryn an Nn values
public static ArrayList<GArea> createWaveGuideInvPhCvary(ArrayList<Double> al, double x, double y, double W, int gdsLayer)

// waveGuideInvPhCvary – al is a Double vector array containing rx1 ry1 N1 .... rxn ryn an Nn values
public static void createWaveGuideInvPhCvary(Struct currentStruct, ArrayList<Double> al, double x, double y, double W, int gdsLayer)
// waveGuidePhC - al is a Double vector array containing rx1 ry1 N1 .... rxn ryn Nn values
public static ArrayList<Area> createWaveGuidePhC(ArrayList<Double> al, double x, double y, double a, double W, double So, double Sh, int gdsLayer)

// waveGuidePhC - al is a Double vector array containing rx1 ry1 N1 .... rxn ryn Nn values
public static void createWaveGuidePhC(Struct currentStruct, ArrayList<Double> al, double x, double y, double a, double W, double So, double Sh, int gdsLayer)

// waveGuidePhCvary - al is a Double vector array containing rx1 ry1 al N1 .... rxn ryn an Nn values
public static ArrayList<Area> createWaveGuidePhCvary(ArrayList<Double> al, double x, double y, double W, double So, double Sh, int gdsLayer)

// waveGuidePhCvary - al is a Double vector array containing rx1 ry1 al N1 .... rxn ryn an Nn values
public static void createWaveGuidePhCvary(Struct currentStruct, ArrayList<Double> al, double x, double y, double W, double So, double Sh, int gdsLayer)

// waveGuideInvRectPhC - al is a Double vector array containing Lx1 Ly1 .... Lxn Lyn values
public static ArrayList<Area> createWaveGuideInvRectPhC(ArrayList<Double> al, double x, double y, double a, double W, int gdsLayer)

// waveGuideInvRectPhC - al is a Double vector array containing Lx1 Ly1 .... Lxn Lyn values
public static void createWaveGuideInvRectPhC(Struct currentStruct, ArrayList<Double> al, double x, double y, double a, double W, int gdsLayer)

// waveGuideInvRectPhCvary - al is a Double vector array containing Lx1 Ly1 .... Lxn Lyn an values
public static ArrayList<Area> createWaveGuideInvRectPhCvary(ArrayList<Double> al, double x, double y, double W, int gdsLayer)

// waveGuideInvRectPhCvary - al is a Double vector array containing Lx1 Ly1 .... Lxn Lyn an values
public static void createWaveGuideInvRectPhCvary(Struct currentStruct, ArrayList<Double> al, double x, double y, double W, int gdsLayer)

// waveGuideRectPhC - al is a Double vector array containing Lx1 Ly1 .... Lxn Lyn values
public static ArrayList<Area> createWaveGuideRectPhC(ArrayList<Double> al, double x, double y, double a, double W, double So, double Sh,
int gdsLayer)

// waveGuideRectPhC − al is a Double vector array containing Lx1 Ly1 . . . Lxn Lyn values
public static void createWaveGuideRectPhC(Struct currentStruct, 
    ArrayList<Double> al, double x, double y, double a, double W, 
    double So, double Sh, int gdsLayer)

// waveGuideRectPhCvary − al is a Double vector array containing Lx1 Ly1 a1 . . . Lxn Lyn a1 values
public static ArrayList<GArea> createWaveGuideRectPhCvary(ArrayList<Double> al, double x, double y, double W, double So, double Sh, int gdsLayer)

// waveGuideRectPhCvary − al is a Double vector array containing Lx1 Ly1 a1 . . . Lxn Lyn an values
public static ArrayList<GArea> createWaveGuideRectPhCvary(Struct currentStruct, 
    ArrayList<Double> al, double x, double y, double W, double So, 
    double Sh, int gdsLayer)

// waveGuideInvRectFlushPhCvary − al is a Double vector array containing Lx1 Ly1 a1 . . . Lxn Lyn an values
public static ArrayList<GArea> createWaveGuideInvRectFlushPhCvary( 
    ArrayList<Double> al, double x, double y, double W, double d, int gdsLayer)

// waveGuideInvRectFlushPhCvary − al is a Double vector array containing Lx1 Ly1 a1 . . . Lxn Lyn an values
public static ArrayList<GArea> createWaveGuideInvRectFlushPhCvary(Struct currentStruct, 
    ArrayList<Double> al, double x, double y, double W, 
    double d, int gdsLayer)

// waveGuideRectFlushPhCvary − al is a Double vector array containing Lx1 Ly1 a1 . . . Lxn Lyn an values
public static ArrayList<GArea> createWaveGuideRectFlushPhCvary( 
    ArrayList<Double> al, double x, double y, double W, double d, 
    double So, double Sh, int gdsLayer)

// waveGuideRectFlushPhCvary − al is a Double vector array containing Lx1 Ly1 a1 . . . Lxn Lyn an values
public static ArrayList<GArea> createWaveGuideRectFlushPhCvary(Struct currentStruct, 
    ArrayList<Double> al, double x, double y, double W, 
    double d, double So, double Sh, int gdsLayer)

// wgcdcdV1
public static GArea createWgcdcd1(double x, double y, double rad, int 
    numSides, double dx, double dy, double wgWidth, double wgLength,
public static GArea createWgdcd2(double x, double y, double rad, int numSides, double dx, double dy, double H2, double W2, double tipH, double tipW, double tipS, double tipY, double wgWidth, double wgLength, double wgRad, double sleeve, double gap, double cutWidth, double cutPosition, double THETA, int gdsLayer)

public static GArea createWgdcd3(double x, double y, double rad, int numSides, double dx, double dy, double dBoxBelowGap, double H2, double W2, double wgWidth, double wgLength, double wgRad, double sleeve, double gap, double cutWidth, double cutPosition, double distance, double THETA, int gdsLayer)

public static GArea createWgdcd4(double x, double y, double rad, int numSides, double dx, double dy, double dBoxBelowGap, double H2, double triH, double wgWidth, double wgLength, double wgRad, double sleeve, double gap, double cutWidth, double cutPosition, double distance, double THETA, int gdsLayer)

public static GArea createWgdcd5(double x, double y, double rad, int numSides, double dx, double dy, double dBoxBelowGap, double triH, double wgWidth, double wgLength, double wgRad, double sleeve, double gap, double cutWidth, double cutPositionUpper, double cutPositionLower, double THETA, int gdsLayer)

public static GArea createWgdcd6(double x, double y, double rad, int numSides, double dx, double dy, double dBoxBelowGap, double triCutH, double triH, double wgWidth, double wgLength, double wgRad, double sleeve, double gap, double cutWidth, double cutPositionUpper, double cutPositionLower, double THETA, int gdsLayer)

public static GArea createWgdcd7(double x, double y, double rad, double rad2, int numSides, double dx, double dy, double wgWidth, double wgLength, double wgRad, double sleeve, double gap, double cutWidth, double cutPosition, double THETA, int gdsLayer)

public static ArrayList<GArea> createWgdcd8(double x, double y, double bottomTaperWidth, double frameWidth, double bottomInteriorWidth, double sleeve, double wgWidthTop, double wgWidthBottom, double wgStraightLengthTop, double wgTaperedLength, double gap, double cutWidth, double cutPosition, double THETA, int gdsLayer)
4.2.9 MEMS NEMS Library Methods

// bentBeam
public static ArrayList<GArea> createBentBeam(double x, double y,
        double width, double length1, double length2, double length3,
        double baseHeight, double baseWidth, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// bentBeam
public static void createBentBeam(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        length3, double baseHeight, double baseWidth, double anchorDistance
        , int anchorLayer, double THETA, int gdsLayer)

// bentBeamArray
public static ArrayList<GArea> createBentBeamArray(double x, double y,
        double width, double length1, double length2, double length3,
        double length4, double hOffset, double pitch, int numElements,
        double centralBeamWidth, double dimpleHeight, double dimpleWidth,
        int dimpleLayer, double baseWidth, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// bentBeamArray
public static void createBentBeamArray(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        length3, double length4, double hOffset, double pitch, int
        numElements, double centralBeamWidth, double dimpleHeight, double
        dimpleWidth, int dimpleLayer, double baseWidth, double
        anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// biMorph
public static ArrayList<GArea> createBiMorph(double x, double y,
        double width1, double width2, double width3, double width4,
        double length1, double length2, double length3, double pitch,
        double dimpleHeight, double dimpleWidth, int dimpleLayer, double
        baseHeight, double baseWidth, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// biMorph
public static void createBiMorph(Struct currentStruct, double x,
        double y, double width1, double width2, double width3, double
        width4, double length1, double length2, double length3, double
        pitch, double dimpleHeight, double dimpleWidth, int dimpleLayer,
        double baseHeight, double baseWidth, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// combDriveV1
public static ArrayList<GArea> createCombDriveV1(double x, double y,
        double width1, double width2, double length1, double length2,
        int numElectrodes, double pitch, double baseHeight, int baseLayer,
        double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
// combDriveV1
public static void createCombDriveV1(Struct currentStruct, double x,
    double y, double width1, double width2, double length1, double
    length2, int numElectrodes, double pitch, double baseHeight, int
    baseLayer, double anchorDistance, int anchorLayer, double THETA,
    int gdsLayer)

// linearDriveV1
public static ArrayList<GArea> createLinearDriveV1(double x, double y,
    double width1, double length1, double length2, double length3,
    double gap, int numElectrodes, double pitch, double baseHeight,
    double baseWidth, double rotorPitch, int rotorLayer, double
    anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// linearDriveV1
public static void createLinearDriveV1(Struct currentStruct, double x,
    double y, double width1, double length1, double length2, double
    length3, double gap, int numElectrodes, double pitch, double
    baseHeight, double baseWidth, double rotorPitch, int rotorLayer,
    double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// foldedSpring1A
public static ArrayList<GArea> createFoldedSpring1A(double x, double y,
    double width, double length1, double length2, double pitch, double
    amplitude, int numberOfPeriods, double baseHeight, double
    baseWidth, double anchorDistance, int anchorLayer, double THETA,
    int gdsLayer)

// foldedSpring1A
public static void createFoldedSpring1A(Struct currentStruct, double x,
    double y, double width, double length1, double length2, double
    pitch, double amplitude, int numberOfPeriods, double baseHeight,
    double baseWidth, double anchorDistance, int anchorLayer, double
    THETA, int gdsLayer)

// foldedSpring2A
public static ArrayList<GArea> createFoldedSpring2A(double x, double y,
    double width, double length1, double length2, double pitch, double
    amplitude, int numberOfPeriods, double baseHeight, double
    baseWidth, double anchorDistance, int anchorLayer, double THETA,
    int gdsLayer)

// foldedSpring2A
public static void createFoldedSpring2A(Struct currentStruct, double x,
    double y, double width, double length1, double length2, double
    pitch, double amplitude, int numberOfPeriods, double baseHeight,
    double baseWidth, double anchorDistance, int anchorLayer, double
    THETA, int gdsLayer)
// foldedSpring1B
public static ArrayList<GArea> createFoldedSpring1B(double x, double y,
        double width, double length1, double length2, double
        amplitude, int numberOfPeriods, double baseHeight, double
        baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring1B
public static void createFoldedSpring1B(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2B
public static ArrayList<GArea> createFoldedSpring2B(double x, double y,
        double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2B
public static void createFoldedSpring2B(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2C
public static ArrayList<GArea> createFoldedSpring2C(double x, double y,
        double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2C
public static void createFoldedSpring2C(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2D
public static ArrayList<GArea> createFoldedSpring2D(double x, double y,
        double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, double baseHeight, double
        baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)
// foldedSpring2D
public static void createFoldedSpring2D(Struct currentStruct, double x, double y, double width, double length1, double length2, double pitch, double amplitude, int numberOfPeriods, double baseHeight, double baseWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// foldedSpring2E
public static ArrayList<GArea> createFoldedSpring2E(double x, double y, double width, double length1, double length2, double pitch, double amplitude, int numberOfPeriods, int numSides, double baseHeight, double baseWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// foldedSpring2F
public static ArrayList<GArea> createFoldedSpring2F(double x, double y, double width, double length1, double length2, double pitch, double amplitude, int numberOfPeriods, int numSides, double baseHeight, double baseWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// foldedSpring2G
public static ArrayList<GArea> createFoldedSpring2G(double x, double y, double width, double length1, double length2, double pitch, double amplitude, int numberOfPeriods, int numSides, double baseHeight, double baseWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
// foldedSpring2H
public static ArrayList<GArea> createFoldedSpring2H(double x, double y,
        double width, double length1, double length2, double pitch, double
        amplitude, int numberOfPeriods, int numSides, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2H
public static void createFoldedSpring2H(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, int numSides, double
        baseHeight, double baseWidth, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// foldedSpring2I
public static ArrayList<GArea> createFoldedSpring2I(double x, double y,
        double width, double length1, double length2, double pitch, double
        amplitude, int numberOfPeriods, int numSides, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2I
public static void createFoldedSpring2I(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, int numSides, double
        baseHeight, double baseWidth, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// foldedSpring2J
public static ArrayList<GArea> createFoldedSpring2J(double x, double y,
        double width, double length1, double length2, double pitch, double
        amplitude, int numberOfPeriods, int numSides, double baseHeight,
        double baseWidth, double anchorDistance, int anchorLayer, double
        THETA, int gdsLayer)

// foldedSpring2J
public static void createFoldedSpring2J(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        pitch, double amplitude, int numberOfPeriods, int numSides, double
        baseHeight, double baseWidth, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// bolometerL
public static ArrayList<GArea> createBolometerL(double x, double y,
        double w1, double w2, double w3, double g1, double g2, double g3,
        double L1, double L2, double r, double ri, int numSides, double a,
        double b, double c, double d, double e, double f, int La, int Lb,
        int Lc, int Ld, int Le, int Lf, double THETA, double shapeReso)
// bolometerL
public static void createBolometerL(Struct currentStruct, double x, 
double y, double w1, double w2, double w3, double g1, double g2, 
double g3, double L1, double L2, double r, double ri, int numSides, 
double a, double b, double c, double d, double e, double f, int La, 
int Lb, int Lc, int Ld, int Le, int Lf, double THETA, double 
shapeReso)

// bolometerU
public static ArrayList<GArea> createBolometerU(double x, double y, 
double w1, double w2, double w3, double g1, double g2, double g3, 
double L1, double L2, double r, double ri, int numSides, double a, 
double b, double c, double d, double e, double f, int La, int Lb, 
int Lc, int Ld, int Le, int Lf, double THETA, double shapeReso)

// gear
public static ArrayList<GArea> createGear(double x, double y, double 
rad, double width, double height, int numberOfGears, int numSides, 
double THETA, int gdsLayer)

// gear
public static void createGear(Struct currentStruct, double x, double y, 
double rad, double width, double height, int numberOfGears, int 
numSides, double THETA, int gdsLayer)

// gearT
public static ArrayList<GArea> createGearT(double x, double y, double 
rad, double width, double height, int numberOfGears, double 
triangleL, int numSides, double THETA, int gdsLayer)

// gearT
public static void createGearT(Struct currentStruct, double x, double y, 
, double rad, double width, double height, int numberOfGears, 
double triangleL, int numSides, double THETA, int gdsLayer)

// straightSpring
public static ArrayList<GArea> createStraightSpring(double x, double y, 
double width, double radiusCenterHub, double widthRing, double 
radiusRing, int numSides, int numElements, double anchorDistance, 
int anchorLayer, double THETA, int gdsLayer)
// createStraightSpring
public static void createStraightSpring(Struct currentStruct, double x, double y, double width, double radiusCenterHub, double widthRing, double radiusRing, int numSides, int numElements, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// createCircularSpring
public static ArrayList<GArea> createCircularSpring(double x, double y, double width, double radiusCenterHub, double widthRing, double radiusRing, int numSides, int numElements, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// createStraightSpringE
public static void createStraightSpringE(Struct currentStruct, double x, double y, double width, double radiusCenterHub, double widthRing, double radiusRing, int numSides, int numElements, double gap, double electrodeWidth, int numElectrodes, double gapFraction, double anchorElectrodeDistance, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// createCircularSpringE
public static ArrayList<GArea> createCircularSpringE(double x, double y, double width, double radiusCenterHub, double widthRing, double radiusRing, int numSides, int numElements, double gap, double electrodeWidth, int numElectrodes, double gapFraction, double anchorElectrodeDistance, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
// combRadialV1
public static ArrayList<GArea> createCombRadialV1(double x, double y,
        double w1, double r1, double w2, double r2, double wc, double gap,
        int numElements, int numSides, double thetaComb, double
        thetaOverlap, double anchorDistance, int anchorLayer, double THETA,
        int gdsLayer)

// combRadialV1
public static void createCombRadialV1(Struct currentStruct, double x,
        double y, double w1, double r1, double w2, double r2, double wc,
        double gap, int numElements, int numSides, double thetaComb, double
        thetaOverlap, double anchorDistance, int anchorLayer, double THETA,
        int gdsLayer)

// combRadialV2
public static ArrayList<GArea> createCombRadialV2(double x, double y,
        double w1, double r1, double w2, double r2, double wc, double gap,
        int numElements, int numSides, double thetaComb, double
        thetaOverlap, double anchorDistance, int anchorLayer, double THETA,
        int gdsLayer)

// combRadialV2
public static void createCombRadialV2(Struct currentStruct, double x,
        double y, double w1, double r1, double w2, double r2, double wc,
        double gap, int numElements, int numSides, double thetaComb, double
        thetaOverlap, double anchorDistance, int anchorLayer, double THETA,
        int gdsLayer)

// flexure2A
public static ArrayList<GArea> createFlexure2A(double x, double y,
        double width, double length1, double length2, double widthMass,
        double lengthMass, double baseHeight, double baseWidth, double
        anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure2A
public static void createFlexure2A(Struct currentStruct, double x,
        double y, double width, double length1, double length2, double
        widthMass, double lengthMass, double baseHeight, double baseWidth,
        double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure2B
public static ArrayList<GArea> createFlexure2B(double x, double y,
        double width, double length1, double length2, double widthMass,
        double lengthMass, double connectorHeight, double connectorWidth,
        double squareHeight, double squareWidth, double baseHeight, double
        baseWidth, double anchorDistance, int anchorLayer, double THETA,
        int gdsLayer)
// flexure2B
public static void createFlexure2B(Struct currentStruct, double x,
double y, double width, double length1, double length2, double
widthMass, double lengthMass, double connectorHeight, double
connectorWidth, double squareHeight, double squareWidth, double
baseHeight, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)

// flexure2C
public static ArrayList<GArea> createFlexure2C(double x, double y,
double width, double length1, double length2, double
length3, double widthMass, double lengthMass, double amplitude, int periods,
double baseHeight, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)

// flexure2C
public static void createFlexure2C(Struct currentStruct, double x,
double y, double width, double length1, double length2, double
length3, double widthMass, double lengthMass, double amplitude, int periods,
double baseHeight, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)

// flexure2D
public static ArrayList<GArea> createFlexure2D(double x, double y,
double width1, double width2, double width3, double width4, double
length1, double length2, double length3, double length4, double
length5, double length6, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)

// flexure2D
public static void createFlexure2D(Struct currentStruct, double x,
double y, double width1, double width2, double width3, double
width4, double length1, double length2, double length3, double
length4, double length5, double length6, double baseWidth, double
anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure2E
public static ArrayList<GArea> createFlexure2E(double x, double y,
double width1, double width2, double width3, double width4, double
width5, double length1, double length2, double length3, double
length4, double gap, double baseExtent, double baseHeight, double
baseWidth, double anchorDistance, int anchorLayer, double THETA, int
anchorLayer, double THETA, int gdsLayer)

// flexure2E
public static void createFlexure2E(Struct currentStruct, double x,
double y, double width1, double width2, double width3, double
width4, double width5, double length1, double length2, double
length3, double length4, double gap, double baseExtent, double
baseHeight, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)
// flexure4A
public static ArrayList<GArea> createFlexure4A(double x, double y,
    double width, double length1, double length2, double widthMass,
    double lengthMass, double baseHeight, double baseExtent, double
    anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure4A
public static void createFlexure4A(Struct currentStruct, double x,
    double y, double width, double length1, double length2, double
    widthMass, double lengthMass, double baseHeight, double baseExtent,
    double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure4B
public static ArrayList<GArea> createFlexure4B(double x, double y,
    double width, double length1, double length2, double widthMass,
    double gap, double baseHeight, double baseWidth, double
    anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure4B
public static void createFlexure4B(Struct currentStruct, double x,
    double y, double width, double length1, double length2, double
    widthMass, double gap, double baseHeight, double baseWidth, double
    anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure4C
public static ArrayList<GArea> createFlexure4C(double x, double y,
    double width, double length1, double length2, double length3,
    double length4, double widthMass, double gap, double baseHeight,
    double baseWidth, double anchorDistance, int anchorLayer, double
    THETA, int gdsLayer)

// flexure4C
public static void createFlexure4C(Struct currentStruct, double x,
    double y, double width, double length1, double length2, double
    length3, double length4, double widthMass, double gap, double
    baseHeight, double baseWidth, double anchorDistance, int
    anchorLayer, double THETA, int gdsLayer)

// flexure4D
public static ArrayList<GArea> createFlexure4D(double x, double y,
    double width1, double width2, double width3, double length1, double
    length2, double widthMass, double lengthMass, double gap, double
    baseHeight, double baseWidth, double anchorDistance, int
    anchorLayer, double THETA, int gdsLayer)
// flexure4D
public static void createFlexure4D(Struct currentStruct, double x,
    double y, double width1, double width2, double width3, double
    length1, double length2, double length3, double widthMass,
    double gap, double baseHeight, double baseWidth, double
    anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// flexure4E
public static ArrayList<Area> createFlexure4E(double x, double y,
    double width1, double width2, double width3, double width4,
    double length1, double length2, double length3, double length4,
    double length5, double baseHeight, double baseWidth,
    double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverL
public static ArrayList<Area> createCantileverL(double x, double y,
    double width, double startL, double pitch, int numElements,
    double baseHeight, double baseExtent, double linearVar,
    double THETA, int gdsLayer)

// cantileverP
public static ArrayList<Area> createCantileverP(double x, double y,
    double width, double startL, double pitch, int numElements,
    double baseHeight, double baseExtent, double percentageVar,
    double THETA, int gdsLayer)

// cantileverSine
public static ArrayList<Area> createCantileverSine(double x, double y,
    double width, double startL, double pitch, int numElements,
    double baseHeight, double baseExtent, double sineAmplitude,
    double THETA, int gdsLayer)
// cantileverSine
public static void createCantileverSine(Struct currentStruct, double x, double y, double width, double startL, double pitch, int numElements, double baseHeight, double baseExtent, double sineAmplitude, double THETA, int gdsLayer)

// cantileverLSE
public static ArrayList<GArea> createCantileverLSE(double x, double y, double width, double startL, double endL, double pitch, int numElements, double baseHeight, double baseExtent, double THETA, int gdsLayer)

// cantileverNLSE
public static void createCantileverNLSE(Struct currentStruct, double x, double y, double width, double startL, double endL, double pitch, int numElements, double baseHeight, double baseExtent, double THETA, int gdsLayer)

// cantileverLSE
public static ArrayList<GArea> createCantileverLSE(double x, double y, double width, double startL, double endL, double pitch, int numElements, double baseHeight, double baseExtent, double THETA, int gdsLayer)

// cantileverNLSE
public static ArrayList<GArea> createCantileverNLSE(Struct currentStruct, double x, double y, double width, double startL, double endL, double pitch, int numElements, double baseHeight, double baseExtent, double THETA, int gdsLayer)

// cantileverCustom - ai is an array list with values s1 w1 L1 .... sn wn Ln
public static ArrayList<GArea> createCantileverCustom(double x, double y, ArrayList<Double> al, double sEnd, double bH, double THETA, int gdsLayer)

// cantileverCustom - ai is an array list with values s1 w1 L1 .... sn wn Ln
public static void createCantileverCustom(Struct currentStruct, double x, double y, ArrayList<Double> al, double sEnd, double bH, double THETA, int gdsLayer)

// cantileverSR
public static ArrayList<GArea> createCantileverSR(double x, double y, double width, double length, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
// cantileverSR
public static void createCantileverSR(Struct currentStruct, double x, double y, double width, double length, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSTri
public static ArrayList<GArea> createCantileverSTri(double x, double y, double width, double length, double triangleHeight, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSTri
public static void createCantileverSTri(Struct currentStruct, double x, double y, double width, double length, double triangleHeight, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSTrapping
public static ArrayList<GArea> createCantileverSTrapping(double x, double y, double widthBottom, double widthTop, double length, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSTrapping
public static void createCantileverSTrapping(Struct currentStruct, double x, double y, double widthBottom, double widthTop, double length, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSPaddle
public static ArrayList<GArea> createCantileverSPaddle(double x, double y, double widthBot, double widthTop, double lengthBot, double lengthTop, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSPaddle
public static void createCantileverSPaddle(Struct currentStruct, double x, double y, double widthBot, double widthTop, double lengthBot, double lengthTop, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSCH
public static ArrayList<GArea> createCantileverSCH(double x, double y, double width, double radius, int numSides, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverSCH
public static void createCantileverSCH(Struct currentStruct, double x, double y, double width, double radius, int numSides, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
// cantileverSCH
public static void createCantileverSCH(Struct currentStruct, double x,
   double y, double width, double radius, int numSides, double baseHeight,
   double baseExtent, double anchorDistance, int anchorLayer, double THETA,
   int gdsLayer)

// cantileverSCF
public static ArrayList<GArea> createCantileverSCF(double x, double y,
   double width, double radius, int numSides, double length, double baseHeight,
   double baseExtent, double anchorDistance, int anchorLayer, double THETA,
   int gdsLayer)

// cantileverHR
public static ArrayList<GArea> createCantileverHR(double x, double y,
   double width, double length, double hollowW, double baseHeight,
   double baseExtent, double anchorDistance, int anchorLayer, double THETA,
   int gdsLayer)

// cantileverHTri
public static ArrayList<GArea> createCantileverHTri(double x, double y,
   double width, double length, double triangleHeight, double hollowW,
   double baseHeight, double baseExtent, double anchorDistance, int anchorLayer,
   double THETA, int gdsLayer)

// cantileverHTrap
public static ArrayList<GArea> createCantileverHTrap(double x, double y,
   double widthBottom, double widthTop, double length, double hollowW,
   double baseHeight, double baseExtent, double anchorDistance, int anchorLayer,
   double THETA, int gdsLayer)
// cantileverHTrap
public static void createCantileverHTrap(Struct currentStruct, double x, double y, double widthBottom, double widthTop, double length, double hollowW, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverHPaddle
public static ArrayList<GArea> createCantileverHPaddle(double x, double y, double widthBot, double widthTop, double lengthTop, double hollowW, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverHCH
public static ArrayList<GArea> createCantileverHCH(double x, double y, double width, double radius, int numSides, double hollowW, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverHCF
public static ArrayList<GArea> createCantileverHCF(double x, double y, double width, double radius, int numSides, double length, double hollowW, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverPB2
public static ArrayList<GArea> createCantileverPB2(double x, double y, double width, double length1, double length2, double lengthTop, double gap, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
// cantileverPB2
public static void createCantileverPB2(Struct currentStruct, double x, double y, double width, double length1, double length2, double lengthTop, double gap, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer) {

// cantileverPB3
public static ArrayList<GArea> createCantileverPB3(double x, double y, double width, double length2, double length3, double lengthTop, double gap, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverPB3
public static void createCantileverPB3(Struct currentStruct, double x, double y, double width, double length1, double length2, double length3, double lengthTop, double gap, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, gdsLayer)

// cantileverUR
public static ArrayList<GArea> createCantileverUR(double x, double y, double width, double length1, double length2, double lengthTop, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverUR
public static void createCantileverUR(Struct currentStruct, double x, double y, double width, double length1, double length2, double lengthTop, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverUCF
public static ArrayList<GArea> createCantileverUCF(double x, double y, double width, double length1, double length2, double lengthTop, double radius, int numSides, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverUCF
public static void createCantileverUCF(Struct currentStruct, double x, double y, double width, double length1, double length2, double lengthTop, double radius, int numSides, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, gdsLayer)

// cantileverUC
public static ArrayList<GArea> createCantileverUC(double x, double y, double width, double length1, double length2, double diameter, int numSides, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
// cantileverUC
public static void createCantileverUC(Struct currentStruct, double x, double y, double width1, double length1, double length2, double diameter, int numSides, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverUCC
public static ArrayList<GArea> createCantileverUCC(double x, double y, double width1, double width2, double length1, double length2, double length3, double lengthTop, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverUCP
public static void createCantileverUCP(Struct currentStruct, double x, double y, double width1, double width2, double length1, double length2, double length3, double length2a, double length2b, double length3, double lengthTop, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverCE
public static void createCantileverCE(Struct currentStruct, double x, double y, double width, double length, double rX, double rY, int numSides, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// cantileverCEPaddle
public static ArrayList<GArea> createCantileverCEPaddle(double x, double y, double width, double length, double rX1, double rY1,
public static void createCantileverCEPaddle (Struct currentStruct, double x, double y, double width, double length, double rX1, double rY1, double rX2, double rY2, int numSides, double paddleW, double paddleL, double baseHeight, double baseExtent, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// createDcBeamL
public static ArrayList<GArea> createDcBeamL (double x, double y, double width, double startL, double pitch, int numElements, double baseHeight, double baseExtent, double linearVar, double THETA, int gdsLayer)

// createDcBeamP
public static ArrayList<GArea> createDcBeamP (double x, double y, double width, double startL, double pitch, int numElements, double baseHeight, double baseExtent, double percentageVar, double THETA, int gdsLayer)

// createDcBeamLSE
public static ArrayList<GArea> createDcBeamLSE (double x, double y, double width, double startL, double endL, double pitch, int numElements, double baseHeight, double baseExtent, double THETA, int gdsLayer)

// createDcBeamP
public static void createDcBeamP (Struct currentStruct, double x, double y, double width, double startL, double pitch, int numElements, double baseHeight, double baseExtent, double percentageVar, double THETA, int gdsLayer)
// dcBeamNLSE
public static ArrayList<GArea> createDcBeamNLSE(double x, double y,
        double width, double startL, double endL, double pitch, int
        numElements, double baseHeight, double baseExtent, double THETA,
        int gdsLayer)

// dcBeamNLSE
public static void createDcBeamNLSE(Struct currentStruct, double x,
        double y, double width, double startL, double endL, double pitch,
        int numElements, double baseHeight, double baseExtent, double THETA,
        int gdsLayer)

// dcBeamCustom – al is a vector array of s1 w1 L1 .... sn wn LN values
public static ArrayList<GArea> createDcBeamCustom(double x, double y,
        ArrayList<Double> al, double sEnd, double bH, double THETA, int
        gdsLayer)

// dcBeamCustom – al is a vector array of s1 w1 L1 .... sn wn LN values
public static void createDcBeamCustom(Struct currentStruct, double x,
        double y, ArrayList<Double> al, double sEnd, double bH, double
        THETA, int gdsLayer)

// dcBeamR
public static ArrayList<GArea> createDcBeamR(double x, double y,
        double width, double length, double baseHeight, double baseExtent,
        double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// dcBeamR
public static void createDcBeamR(Struct currentStruct, double x, double
        y, double width, double length, double baseHeight, double
        baseExtent, double anchorDistance, int anchorLayer, double THETA,
        int gdsLayer)

// dcBeamT
public static ArrayList<GArea> createDcBeamT(double x, double y,
        double width1, double length1, double width2, double length2,
        double baseHeight, double baseExtent, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// dcBeamT
public static void createDcBeamT(Struct currentStruct, double x, double
        y, double width1, double length1, double width2, double length2,
        double baseHeight, double baseExtent, double anchorDistance, int
        anchorLayer, double THETA, int gdsLayer)

// dcBeamT2
public static ArrayList<GArea> createDcBeamT2(double x, double y,
        double width1, double width2, double width3, double width4, double
        length1, double length2, double length3, double gap, double
baseHeight, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)

// dcBeamT2
public static void createDcBeamT2(Struct currentStruct, double x,
double y, double width1, double width2, double width3, double
width4, double length1, double length2, double length3, double gap,
double baseHeight, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)

// dcBeamCB
public static ArrayList<GArea> createDcBeamCB(double x, double y,
double width, double lengthStart, double lengthEnd, int numElements,
double baseHeight, double baseWidth, double anchorDistance, int
anchorLayer, double THETA, int gdsLayer)

// dcBeamC
public static ArrayList<GArea> createDcBeamC(double x, double y,
double width, double length, double rX1, double rY1, double rX2,
double rY2, int numSides, double baseHeight, double baseExtent,
double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// MARAs
public static void createMARAs(Struct currentStruct, String id, double
x, double y, int numElements, double L1, double W1, double L2,
double W2, double space, double hOverlap, double lengthSide, double
hElectrode, double LB, double HB, double diameter, int numSides,
int layerFront, int layerBack, int layerMetal, double THETA)

// MARA
public static ArrayList<GArea> createMARA(double x, double y, int
numElements, double L1, double W1, double L2, double W2, double space,
double hOverlap, double lengthSide, double hElectrode,
double LB, double HB, double diameter, int numSides, int layerFront,
int layerBack, int layerMetal, double THETA)
// MARA
public static void createMARA(Struct currentStruct, double x, double y,
    int numElements, double L1, double W1, double L2, double W2,
    double space, double hOverlap, double lengthSide, double hElectrode,
    double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MATALWs
public static void createMATALWs(Struct currentStruct, String id,
    double x, double y, int numElements, double L, double Wa, double Wb,
    double space, double hOverlap, double lengthSide, double hElectrode,
    double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MATALW
public static ArrayList<GArea> createMATALW(double x, double y, int numElements, double L, double Wa, double Wb,
    double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB,
    double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MATAs
public static void createMATAs(Struct currentStruct, String id, double x, double y, int numElements, double L1, double Wa, double W1a, double W1b,
    double L2, double W2a, double W2b, double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB,
    double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MATA
public static ArrayList<GArea> createMATA(double x, double y, int numElements, double L1, double Wa, double W1a, double W1b,
    double L2, double W2a, double W2b, double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB,
    double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)
// MAR2s
public static void createMAR2s(Struct currentStruct, String id, double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double lengthSide, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MAR2
public static ArrayList<Area> createMAR2(double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double lengthSide, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MAR3s
public static void createMAR3s(Struct currentStruct, String id, double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double lengthSide, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MAR3
public static ArrayList<Area> createMAR3(double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double lengthSide, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)
// MARCs
public static void createMARCs(Struct currentStruct, String id, double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MARC
public static ArrayList<GArea> createMARC(double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MAT2s
public static void createMAT2s(Struct currentStruct, String id, double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double lengthSide, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MAT2
public static ArrayList<GArea> createMAT2(double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double lengthSide, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MAT2
public static void createMAT2(Struct currentStruct, double x, double y, int numElements, double L1, double W1a, double W1b, double L2, double W2a, double W2b, double space, double lowerSpace, double hOverlap, double hElectrode, double lengthSide, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, double THETA)

// MAT3s
public static void createMAT3s(Struct currentStruct, String id, double x, double y, int numElements, double L1, double W1a, double W1b,
double L2, double W2a, double W2b, double space, double lowerSpace, 
double hOverlap, double hElectrode, double LB, double HB, double 
diameter, int numSides, int layerFront, int layerBack, int 
layerMetal, double THETA)

// MAT3
public static ArrayList<GArea> createMAT3(double x, double y, int 
umElements, double L1, double W1a, double W1b, double L2, double 
W2a, double W2b, double space, double lowerSpace, double hOverlap, 
double hElectrode, double LB, double HB, double diameter, int 
umSides, int layerFront, int layerBack, int layerMetal, double 
THETA)

// MAT3
public static void createMAT3(Struct currentStruct, double x, double y, 
int numElements, double L1, double W1a, double W1b, double L2, 
double W2a, double W2b, double space, double lowerSpace, double 
hOverlap, double hElectrode, double LB, double HB, double diameter, int 
umSides, int layerFront, int layerBack, int layerMetal, double 
THETA)

// MATCs
public static void createMATCs(Struct currentStruct, String id, double 
x, double y, int numElements, double L1, double W1a, double W1b, 
double L2, double W2a, double W2b, double space, double lowerSpace, 
double hOverlap, double hElectrode, double LB, double HB, double 
diameter, int numSides, int layerFront, int layerBack, int layerMetal, 
double THETA)

// MATC
public static ArrayList<GArea> createMATC(double x, double y, int 
umElements, double L1, double W1a, double W1b, double L2, double 
W2a, double W2b, double space, double lowerSpace, double hOverlap, 
double hElectrode, double LB, double HB, double diameter, int 
umSides, int layerFront, int layerBack, int layerMetal, double 
THETA)

// MATC
public static void createMATC(Struct currentStruct, double x, double y, 
int numElements, double L1, double W1a, double W1b, double L2, 
double W2a, double W2b, double space, double lowerSpace, double 
hOverlap, double hElectrode, double LB, double HB, double diameter, int 
umSides, int layerFront, int layerBack, int layerMetal, double 
THETA)

// MARALINEARs
public static void createMARALINEARs(Struct currentStruct, String id, 
double x, double y, int numElements, double L, double dL, double W, 
double space, double hOverlap, double lengthSide, double 
hElectrode, double LB, double HB, double diameter, int numSides, 
int layerFront, int layerBack, int layerMetal, double THETA)
// MARALINEAR
public static ArrayList<GArea> createMARALINEAR(double x, double y, int numElements, double L, double dL, double W, double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA)

// MARALINEAR
public static void createMARALINEAR(Struct currentStruct, double x, double y, int numElements, double L, double dL, double W, double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal)

// MARACURVES
public static void createMARACURVES(Struct currentStruct, String id, double x, double y, int numElements, double L, double dL, double W, double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double shapeReso)

// MARACURVE
public static ArrayList<GArea> createMARACURVE(double x, double y, int numElements, double L, double dL, double W, double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA, double shapeReso)

// MARACURVE
public static void createMARACURVE(Struct currentStruct, double x, double y, int numElements, double L, double dL, double W, double space, double hOverlap, double lengthSide, double hElectrode, double LB, double HB, double diameter, int numSides, int layerFront, int layerBack, int layerMetal, double THETA, double shapeReso)

// GuckelRing
public static ArrayList<GArea> createGuckelRing(double x, double y, double ringWidth, double radius, int numOfSides, double beamWidth, double base, double connectorLegth, double connectorWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

// GuckelRing
public static void createGuckelRing(Struct currentStruct, double x, double y, double ringWidth, double radius, int numOfSides, double beamWidth, double base, double connectorLegth, double connectorWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)
public static ArrayList<GArea> createGuckelRingArray(double x, double y, double ringWidth, double radiusStart, double radiusEnd, double deltaRadius, int numOfSides, double beamWidth, double base, double connectorLength, double connectorWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

public static void createGuckelRingArray(Struct currentStruct, double x, double y, double ringWidth, double radiusStart, double radiusEnd, double deltaRadius, int numOfSides, double beamWidth, double base, double connectorLength, double connectorWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

public static ArrayList<GArea> createDiamondRing(double x, double y, double width1, double width2, double width3, double length1, double length2, double length3, double baseHeight, double baseWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

public static void createDiamondRing(Struct currentStruct, double x, double y, double width1, double width2, double width3, double length1, double length2, double length3, double baseHeight, double baseWidth, double anchorDistance, int anchorLayer, double THETA, int gdsLayer)

public static ArrayList<GArea> createFluidCell(double x, double y, double w1, double w2, double w2T, double w3, double w4, double r1, double L1, double L2, double r, double r1, double r2, double r3, int numSides, double a, double b, double c, double d, double e, double f, double g, double h, double i, int La, int Lb, int Lc, int Ld, int Le, int Lf, double THETA, double shapeReso)

public static void createFluidCell(Struct currentStruct, double x, double y, double w1, double w2, double w2T, double w3, double w4, double r1, double L1, double L2, double r, double r1, double r2, double r3, int numSides, double a, double b, double c, double d, double e, double f, double g, double h, double i, int La, int Lb, int Lc, int Ld, int Le, int Lf, double THETA, double shapeReso)


[77] http://www.genisysgmbh.de/web/applications.html, “GenISys - 3-D e-Beam Lithography Application Note,”


[88] K. Najafi and K. Suzuki, “A novel technique and structure for the measurement of intrinsic stress and Young’s modulus of thin films,” in *IEEE*


