

Contents

VERSION 2019.05.01

Email: Nanolithography.Toolbox@nist.gov

0.1	Nanophotonics - CNST Special Scripts	5
0.1.1	Couplers	5
0.1.1.1	Directional Couplers Inverse	5
0.1.1.2	Multimode Interference Couplers	9
0.1.2	Disc-Ring Architectures	11
0.1.2.1	Disc-Ring Infinite	11
0.1.2.2	Disc-Ring Symmetric Arc	13
0.1.2.3	Disc-Ring Symmetric Inverse Arc	15
0.1.2.4	Disc-Ring Symmetric Inverse Positive Tone Arc	17
0.1.2.5	Disc-Ring Pulley Arc	19
0.1.2.6	Disc-Ring Pulley Inverse Arc	21
0.1.2.7	Disc-Ring Pulley Inverse Positive Tone Arc	23
0.1.3	Waveguides	25
0.1.3.1	180 degree bend with taper	25
0.1.3.2	Racetrack-Waveguides	26
0.1.3.3	Racetrack-Waveguides Inverse	27
0.1.3.4	Racetrack-Waveguides Positive Inverse	28
0.1.3.5	Racetrack-Waveguides Symmetric	29
0.1.3.6	Racetrack-Waveguides Symmetric Inverse	30
0.1.3.7	Racetrack-Waveguides Symmetric Positive Inverse	31
0.1.3.8	Waveguide with linear taper	32
0.1.3.9	Waveguide Tapers With Supports	33
0.1.3.10	Waveguide Wave	35
0.1.3.11	Waveguide Wave Array	36
0.2	Arrays and Instances	37
0.2.1	Rectangular Arrays	37
0.2.2	Rectangular Array - Centered	38
0.2.3	Hexagonal Arrays	39
0.2.4	Hexagonal Array - Centered	40
0.3	Text Labels, PostScript and Logos	41
0.3.1	Text	41
0.3.2	Text - Outline	42
0.3.3	Text - Fill	43
0.3.4	Text - Outline Fill	44
0.3.5	Text - Dashed	45
0.3.6	Label Maker	46
0.3.7	Label Maker - Outline Text	48
0.3.8	Label Maker - Fill Text	50
0.3.9	Label Maker - Outline Fill Text	52

Chapter 0 CONTENTS

0.3.10	Label Maker - Dashed Text	54
0.3.11	CNST and NIST logos	56
0.3.12	CNST and NIST logos - Outline and Dashed	57
0.4	Objects - CNST Special Scripts	58
0.4.1	Archimedes spiral with skipped turns	58
0.4.2	Crossing wires	59
0.4.3	Elliptical Torus-Arc	62
0.4.4	Junctions	63
0.4.4.1	Cross Junction	63
0.4.4.2	Funnel	66
0.4.4.3	Funnel Junction	67
0.4.5	Radial Fill Along An Arc	70
0.4.6	Radially distributed circles of varying radii	71
0.4.7	Radial Arrays	74
0.4.7.1	Variable ring position and shape pitch	75
0.4.7.2	Constant ring pitch	77
0.4.7.3	Linearly varying ring separation	78
0.4.7.4	Exponentially varying ring separation	80
0.4.8	Random Radial Arrays with SVG Export	82
0.4.9	Non-uniform Arrays	85
0.4.9.1	Generalized non-uniform array	86
0.4.9.2	Linearly varying pitch array	87
0.4.9.3	Linearly varying pitch symmetric array	89
0.4.9.4	Linearly varying pitch aligned array	93
0.4.9.5	Linearly varying pitch aligned symmetric array	95
0.4.9.6	Linearly varying vertical and constant horizontal pitch	99
0.4.9.7	Linearly varying vertical and constant horizontal pitch, aligned array	101
0.4.9.8	Linearly varying vertical and constant horizontal pitch symmetric array	103
0.4.9.9	Linearly varying vertical and constant horizontal pitch, aligned symmetric array	107
0.4.9.10	Exponentially varying pitch array	111
0.4.9.11	Exponentially varying pitch - aligned	112
0.4.9.12	Exponentially varying pitch symmetric array	113
0.4.9.13	Exponentially varying pitch, aligned symmetric array	114
0.4.9.14	Exponentially varying vertical and constant horizontal pitch array	115
0.4.9.15	Exponentially varying vertical and constant horizontal pitch, aligned array	116
0.4.9.16	Exponentially varying vertical and constant horizontal pitch symmetric array	117
0.4.9.17	Exponentially varying vertical and constant horizontal pitch, symmetric aligned array	118

Chapter 0 CONTENTS

0.4.10	Paths and Shapes With Splines	119
0.4.11	Resolution Test Pattern	122
0.4.12	Rounded Paths and Shapes	123
0.4.13	Spin-Ice Lattice	124
0.4.14	Van der Pauw Structures	126
0.4.15	Wire Contacts	128
0.5	Shapes - CNST Special Scripts	129
0.5.1	Circles and Ellipses	129
0.5.1.1	Primitive Circle and Ellipse	129
0.5.1.2	Vectorized Circle and Ellipse	129
0.5.1.3	Primitive Circle and Ellipse Outline	130
0.5.1.4	Vectorized Circle and Ellipse Outline	130
0.5.1.5	Primitive Circle and Ellipse Dashed	131
0.5.1.6	Vectorized Circle and Ellipse Dashed	131
0.5.2	Dashed Structures - Cap and Join	132
0.5.2.1	End-Cap and Line-Join Parameters	132
0.5.3	Cross Shapes	133
0.5.3.1	Cross	133
0.5.3.2	Cross - Outline and Dashed	134
0.5.3.3	Parameterized Cross	135
0.5.3.4	Parameterized Cross - Outline and Dashed	136
0.5.3.5	Rounded Cross	137
0.5.3.6	Rounded Cross - Outline and Dashed	138
0.5.3.7	Parameterized Rounded Cross	139
0.5.3.8	Parameterized Rounded Cross - Outline and Dashed	140
0.5.4	L-Shapes	141
0.5.4.1	L-Shape	141
0.5.4.2	L-Shape - Outline and Dashed	142
0.5.4.3	L-Shape Round	143
0.5.4.4	L-Shape Round - Outline and Dashed	144
0.5.5	Rectangular and Square Shapes	145
0.5.5.1	Rectangle	145
0.5.5.2	Rectangle Outline	146
0.5.5.3	Rectangle Dashed	147
0.5.5.4	Square	148
0.5.5.5	Square Outline	149
0.5.5.6	Square Dashed	150
0.5.5.7	Rounded Rectangle	151
0.5.5.8	Rounded Rectangle Outline	152
0.5.5.9	Rounded Rectangle Dashed	153
0.5.5.10	Rounded Square	154
0.5.5.11	Rounded Square Outline	155
0.5.5.12	Rounded Square Dashed	156
0.6	Alignment Elements - CNST Special Scripts	157
0.6.1	Box in box alignment marks	157
0.6.2	Arrows	158

0.6.2.1	Arrow Shapes	158
0.6.2.2	Arrow Shapes - Outline and Dashed	159
0.6.3	Axes	160
0.6.3.1	Axes	160
0.6.3.2	Axes - Outline	161
0.6.3.3	Axes - Dashed	162
0.7	MEMS-NEMS - CNST Special Scripts	163
0.7.1	Beam With Curved Ends	163
0.7.2	Bolometers	164
0.7.3	Suspended Ring Array	165
0.8	General Area Operations	168
0.8.1	General Area - Outline	168
0.9	Interface Functions	170
0.9.1	Reading-Importing GDS Files	170
0.9.2	Reading-Importing and Parsing GDS Files	171
0.9.3	Exporting shape vertices - POINTSXY	172
0.9.4	Shape Resolution	175
0.9.5	Font Outline Width	175
0.10	Path Length Calculations	176
0.10.1	Path Length Calculations of NanoPhotonic Elements	176
0.11	CNST Special Reference Methods	178
0.11.1	CNST Special - Miscellaneous Object Methods	178

0.1 Nanophotonics - CNST Special Scripts

0.1.1 Couplers

0.1.1.1 Directional Couplers Inverse

`<x y L1 w1 L2 L3 w2 g r Nsides $\theta_{(x,y)}$ directionalCoupler1i>`

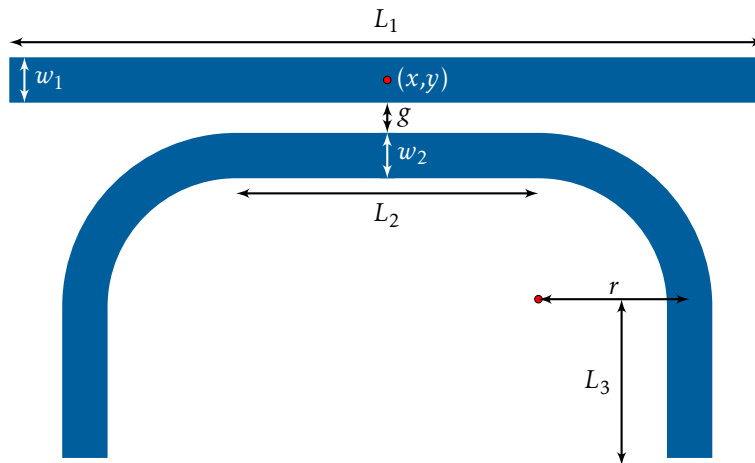


Figure 1: Directional coupler example using the `directionalCoupler1i` constructor.

$\langle x \ y \ L_1 \ w_1 \ L_2 \ L_3 \ w_2 \ g \ r \ N_{sides} \ \theta_{(x,y)} \ \text{directionalCoupler2i} \rangle$

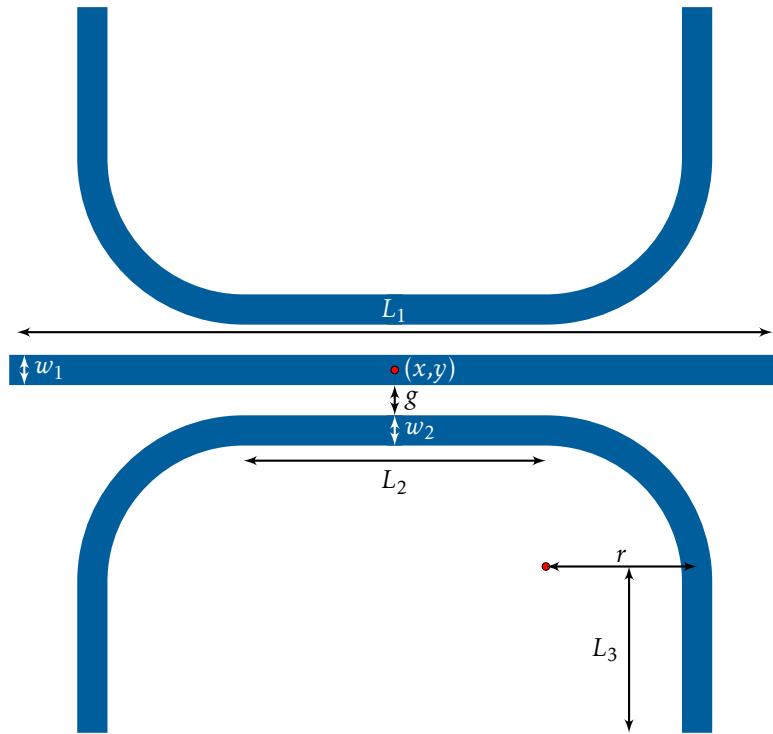


Figure 2: Directional coupler example using the `directionalCoupler2i` constructor.

$\langle x \ y \ w \ g \ L_1 \ L_2 \ r \ N_{sides} \ \theta_{(x,y)} \ \text{directionalCoupler3i} \rangle$

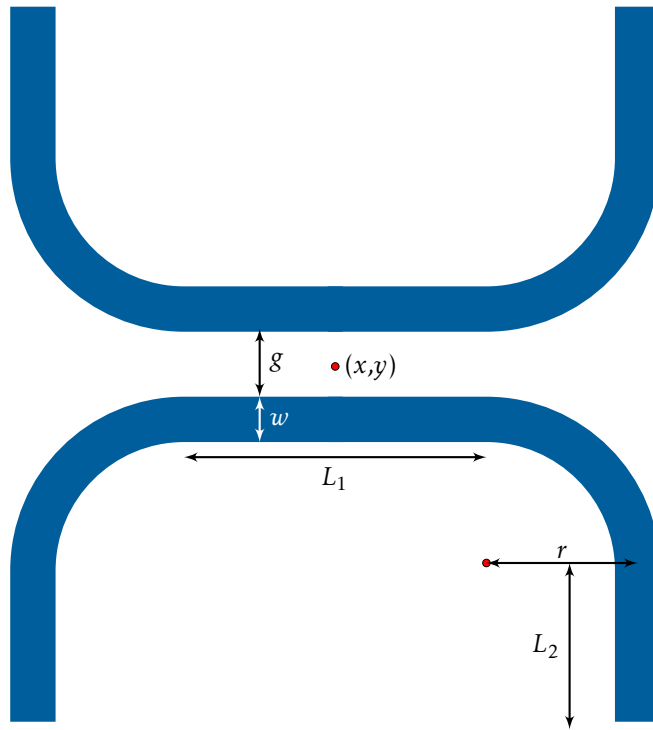


Figure 3: Directional coupler example using the [directionalCoupler3i](#) constructor.

$\langle x \ y \ w \ g \ L_1 \ L_B \ H_B \ \theta_{(x,y)} \ \text{directionalCoupler4i} \rangle$

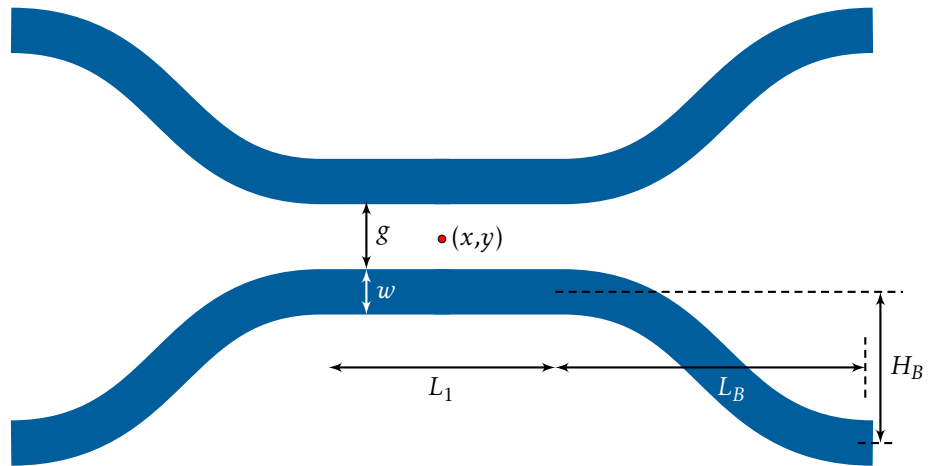


Figure 4: Directional coupler example using the `directionalCoupler4i` constructor.

0.1.1.2 Multimode Interference Couplers

Below are 1×2 multimode interference coupler designs with and without tapered inputs and outputs.

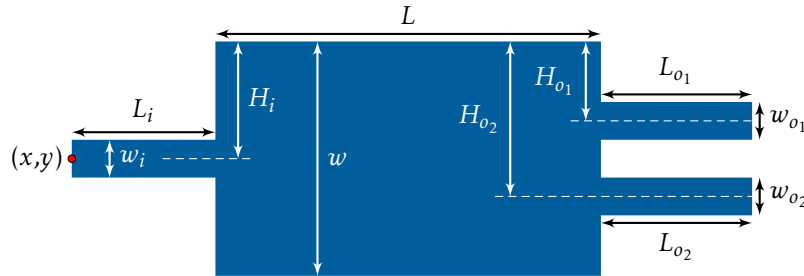


Figure 5: Tapered 1×2 multimode interference coupler.

x y w_i L_i H_i w L w_{o1} L_{o1} H_{o1} w_{o2} L_{o2} H_{o2} $\theta_{(x,y)}$ 1X2MMI

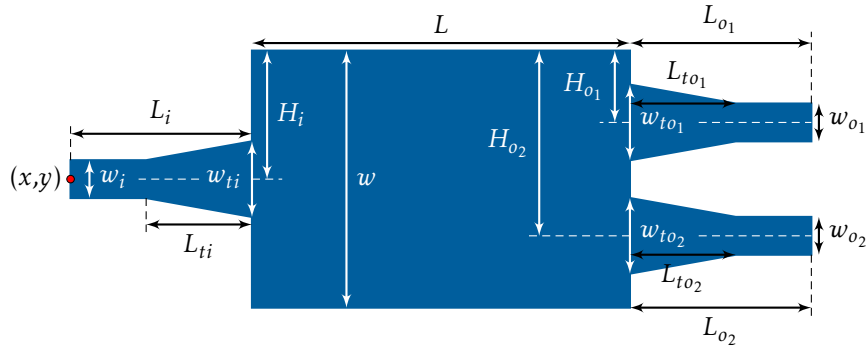


Figure 6: 1×2 multimode interference coupler with tapers.

x y w_i L_i w_{ti} L_{ti} H_i w L w_{o1} L_{o1} w_{to1} L_{to1} H_{o1} w_{o2} L_{o2} w_{to2} L_{to2} H_{o2} $\theta_{(x,y)}$ 1X2MMItaper

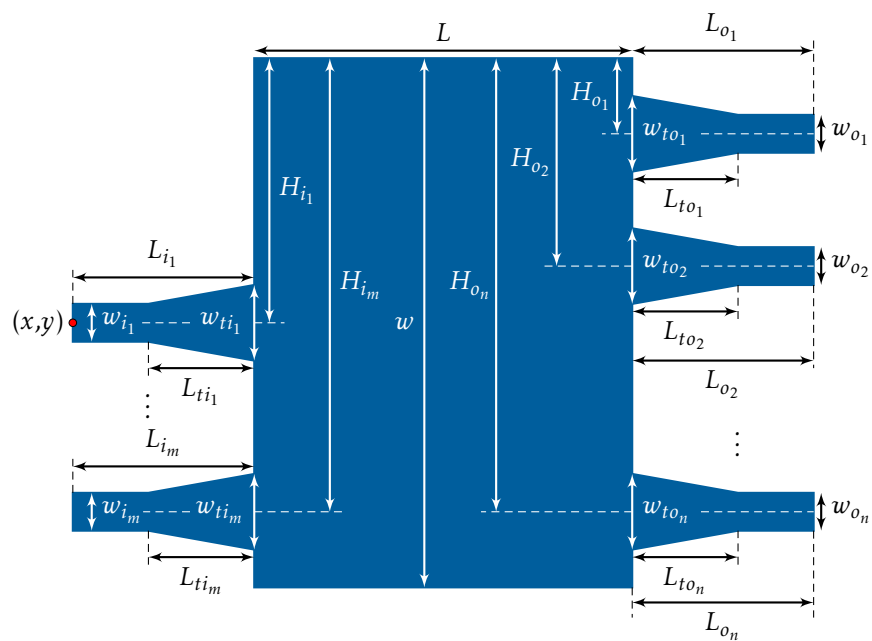


Figure 7: $m \times n$ multimode interference coupler with tapers.

x y w_{i_1} L_{i_1} w_{ti_1} L_{ti_1} H_{i_1} \dots w_{i_m} L_{i_m} w_{ti_m} L_{ti_m} H_{i_m} w L
 w_{o_1} L_{o_1} w_{to_1} L_{to_1} H_{o_1} \dots w_{o_n} L_{o_n} w_{to_n} L_{to_n} H_{o_n} m n $\theta_{(x,y)}$ **mXnMMItaper**

0.1.2 Disc-Ring Architectures

0.1.2.1 Disc-Ring Infinite

$\langle x \ y \ r_d \quad N \ g \ L \ W \ EC \ \text{discInfinite} \rangle$

$\langle x \ y \ r_d \quad \quad \quad g \ L \ W \ EC \ \text{discInfiniteV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g \ L \ W \ EC \ \text{ringInfinite} \rangle$

$\langle x \ y \ r_i \ W_r \quad \quad \quad g \ L \ W \ EC \ \text{ringInfiniteV} \rangle$

$\langle x \ y \ r_d \quad N \ g_1 \ L_1 \ W_1 \ g_2 \ L_2 \ W_2 \ EC \ \text{discInfDS} \rangle$

$\langle x \ y \ r_d \quad \quad \quad g_1 \ L_1 \ W_1 \ g_2 \ L_2 \ W_2 \ EC \ \text{discInfDSV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_1 \ W_1 \ g_2 \ L_2 \ W_2 \ EC \ \text{ringInfDS} \rangle$

$\langle x \ y \ r_i \ W_r \quad \quad \quad g_1 \ L_1 \ W_1 \ g_2 \ L_2 \ W_2 \ EC \ \text{ringInfDSV} \rangle$

$\langle x \ y \ r_d \quad N \ g \ L \ W \ W_e \ EC \ \text{discInfiniteInv} \rangle$

$\langle x \ y \ r_d \quad \quad \quad g \ L \ W \ W_e \ EC \ \text{discInfiniteInvV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g \ L \ W \ W_e \ EC \ \text{ringInfiniteInv} \rangle$

$\langle x \ y \ r_i \ W_r \quad \quad \quad g \ L \ W \ W_e \ EC \ \text{ringInfiniteInvV} \rangle$

$\langle x \ y \ r_d \quad N \ g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{discInflInvDS} \rangle$

$\langle x \ y \ r_d \quad \quad \quad g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{discInflInvDSV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{ringInflInvDS} \rangle$

$\langle x \ y \ r_i \ W_r \quad \quad \quad g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{ringInflInvDSV} \rangle$

$\langle x \ y \ r_d \quad r_e \ N \ g \ L \ W \ W_e \ EC \ \text{discInfiniteInvPos} \rangle$
 $\langle x \ y \ r_d \quad r_e \quad g \ L \ W \ W_e \ EC \ \text{discInfiniteInvPosV} \rangle$
 $\langle x \ y \ r_i \ W_r \ r_e \ N \ g \ L \ W \ W_e \ EC \ \text{ringInfiniteInvPos} \rangle$
 $\langle x \ y \ r_i \ W_r \ r_e \quad g \ L \ W \ W_e \ EC \ \text{ringInfiniteInvPosV} \rangle$

$\langle x \ y \ r_d \quad r_e \ N \ g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{discInflInvPosDS} \rangle$
 $\langle x \ y \ r_d \quad r_e \quad g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{discInflInvPosDSV} \rangle$
 $\langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{ringInflInvPosDS} \rangle$
 $\langle x \ y \ r_i \ W_r \ r_e \quad g_1 \ L_1 \ W_1 \ W_{e1} \ g_2 \ L_2 \ W_2 \ W_{e2} \ EC \ \text{ringInflInvPosDSV} \rangle$

0.1.2.2 Disc-Ring Symmetric Arc

<x y r_d N g θ N_{wg} W L_S EC [discSymmetricA](#)>

<x y r_d g θ W L_S EC [discSymmetricAV](#)>

<x y r_i W_r N g θ N_{wg} W L_S EC [ringSymmetricA](#)>

<x y r_i W_r g θ W L_S EC [ringSymmetricAV](#)>

<x y r_d N g₁ θ N_{wg} W₁ L_S g₂ W₂ L₂ EC [discSymADS](#)>

<x y r_d g₁ θ W₁ L_S g₂ W₂ L₂ EC [discSymADSV](#)>

<x y r_d N g₁ θ₁ N_{wg} W₁ L_{S1} g₂ θ₂ W₂ L_{S2} EC [discSymAPul](#)>

<x y r_d g₁ θ₁ W₁ L_{S1} g₂ θ₂ W₂ L_{S2} EC [discSymAPulV](#)>

<x y r_i W_r N g₁ θ N_{wg} W₁ L_S g₂ W₂ L₂ EC [ringSymADS](#)>

<x y r_i W_r g₁ θ W₁ L_S g₂ W₂ L₂ EC [ringSymADSV](#)>

<x y r_i W_r N g θ₁ N_{wg} W L_S g₂ θ₂ W₂ L_{S2} EC [ringSymAPul](#)>

<x y r_i W_r g θ₁ W L_S g₂ θ₂ W₂ L_{S2} EC [ringSymAPulV](#)>

<x y r_d N g L_c N_{wg} W L_S EC [discSymmetricLCA](#)>

<x y r_d g L_c N_{wg} W L_S EC [discSymmetricLCAV](#)>

<x y r_i W_r N g L_c N_{wg} W L_S EC [ringSymmetricLCA](#)>

<x y r_i W_r g L_c W L_S EC [ringSymmetricLCAV](#)>

<x y r_d N g₁ L_c N_{wg} W₁ L_S g₂ W₂ L₂ EC [discSymLCADS](#)>

<x y r_d g₁ L_c W₁ L_S g₂ W₂ L₂ EC [discSymLCADSV](#)>

$\langle x \ y \ r_d \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ L_{S2} \ EC \ \text{discSymLCAPul} \rangle$

$\langle x \ y \ r_d \ \quad g_1 \ L_{c1} \quad \quad W_1 \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ L_{S2} \ EC \ \text{discSymLCAPulV} \rangle$

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

$\langle x \ y \ r_i \ W_r \quad g_1 \ L_c \quad \quad W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ \text{ringSymLCADSV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ L_{S2} \ EC \ \text{ringSymLCAPul} \rangle$

$\langle x \ y \ r_i \ W_r \quad g_1 \ L_{c1} \quad \quad W_1 \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ L_{S2} \ EC \ \text{ringSymLCAPulV} \rangle$

0.1.2.3 Disc-Ring Symmetric Inverse Arc

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

<x y r_d g θ W W_e L_S EC discSymmetricInvAV>

<x y r_i W_r N g θ N_{wg} W W_e L_S EC ringSymmetricInvA>

<x y r_i W_r g θ W W_e L_S EC ringSymmetricInvAV>

<x y r_d N g_1 θ N_{wg} W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC discSymInvADS>

<x y r_d g_1 θ W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC discSymInvADSV>

<x y r_d N g_1 θ_1 N_{wg} W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_{S2} EC discSymInvAPul>

<x y r_d g_1 θ_1 W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_{S2} EC discSymInvAPulV>

<x y r_i W_r N g_1 θ N_{wg} W_1 W_e L_S g_2 W_2 W_{e2} L_2 EC ringSymInvADS>

<x y r_i W_r g_1 θ W_1 W_e L_S g_2 W_2 W_{e2} L_2 EC ringSymInvADSV>

<x y r_i W_r N g_1 θ_1 N_{wg} W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_{S2} EC ringSymInvAPul>

<x y r_i W_r g_1 θ_1 W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_{S2} EC ringSymInvAPulV>

<x y r_d N g L_c N_{wg} W W_e L_S EC discSymmetricInvLCA>

<x y r_d g L_c W W_e L_S EC discSymmetricInvLCAV>

<x y r_i W_r N g L_c N_{wg} W W_e L_S EC ringSymmetricInvLCA>

<x y r_i W_r g L_c W W_e L_S EC ringSymmetricInvLCAV>

<x y r_d N g_1 L_c N_{wg} W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC discSymInvLCADS>

<x y r_d g_1 L_c W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC discSymInvLCADSV>

$\langle x \ y \ r_d \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{discSymInvLCAPul} \rangle$

$\langle x \ y \ r_d \ \quad g_1 \ L_c \quad \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{discSymInvLCAPulV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringSymInvLCADS} \rangle$

$\langle x \ y \ r_i \ W_r \ \quad g_1 \ L_c \quad \quad W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringSymInvLCADSV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringSymInvLCAPul} \rangle$

$\langle x \ y \ r_i \ W_r \ \quad g_1 \ L_{c1} \quad \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringSymInvLCAPulV} \rangle$

0.1.2.4 Disc-Ring Symmetric Inverse Positive Tone Arc

<x y r_d r_e N g θ N_{wg} W W_e L_S EC discSymmetricInvPosA>

<x y r_d r_e g θ W W_e L_S EC discSymmetricInvPosAV>

<x y r_i W_r r_e N g θ N_{wg} W W_e L_S EC ringSymmetricInvPosA>

<x y r_i W_r r_e g θ W W_e L_S EC ringSymmetricInvPosAV>

<x y r_d r_e N g₁ θ N_{wg} W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discSymInvPosADS>

<x y r_d r_e g₁ θ W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discSymInvPosADSV>

<x y r_d r_e N g₁ θ₁ N_{wg} W₁ W_{e1} L_{S1} g₂ θ₂ W₂ W_{e2} L_{S2} EC discSymInvPosAPul>

<x y r_d r_e g₁ θ₁ W₁ W_{e1} L_{S1} g₂ θ₂ W₂ W_{e2} L_{S2} EC discSymInvPosAPulV>

<x y r_i W_r r_e N g θ N_{wg} W W_e L_S g₂ W₂ W_{e2} L₂ EC ringSymInvPosADS>

<x y r_i W_r r_e g θ W W_e L_S g₂ W₂ W_{e2} L₂ EC ringSymInvPosADSV>

<x y r_i W_r r_e N g₁ θ₁ N_{wg} W₁ W_{e1} L_{S1} g₂ θ₂ W₂ W_{e2} L_{S2} EC ringSymInvPosAPul>

<x y r_i W_r r_e g₁ θ₁ W₁ W_{e1} L_{S1} g₂ θ₂ W₂ W_{e2} L_{S2} EC ringSymInvPosAPulV>

<x y r_d r_e N g L_c N_{wg} W W_e L_S EC discSymmetricInvPosLCA>

<x y r_d r_e g L_c W W_e L_S EC discSymmetricInvPosLCAV>

<x y r_i W_r r_e N g L_c N_{wg} W W_e L_S EC ringSymmetricInvPosLCA>

<x y r_i W_r r_e g L_c W W_e L_S EC ringSymmetricInvPosLCAV>

<x y r_d r_e N g₁ L_c N_{wg} W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discSymInvPosLCADS>

<x y r_d r_e g₁ L_c W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discSymInvPosLCADSV>

$\langle x \ y \ r_d \ r_e \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{discSymInvPosLCAPul} \rangle$

$\langle x \ y \ r_d \ r_e \ \quad g_1 \ L_{c1} \quad \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{discSymInvPosLCAPulV} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringSymInvPosLCADS} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ \quad g_1 \ L_c \quad \quad W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringSymInvPosLCADSV} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringSymInvPosLCAPul} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ \quad g_1 \ L_{c1} \quad \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringSymInvPosLCAPulV} \rangle$

0.1.2.5 Disc-Ring Pulley Arc

<x y r_d N g θ N_{wg} W L_S EC [discPulleyA](#)>

<x y r_d g θ W L_S EC [discPulleyAV](#)>

<x y r_i W_r N g θ N_{wg} W L_S EC [ringPulleyA](#)>

<x y r_i W_r g θ W L_S EC [ringPulleyAV](#)>

<x y r_d N g₁ θ N_{wg} W₁ L_S g₂ W₂ L₂ EC [discPulADS](#)>

<x y r_d g₁ θ W₁ L_S g₂ W₂ L₂ EC [discPulADSV](#)>

<x y r_d N g₁ θ₁ N_{wg} W₁ L_{S1} g₂ θ₂ W₂ L_{S2} EC [discPulAPul](#)>

<x y r_d g₁ θ₁ W₁ L_{S1} g₂ θ₂ W₂ L_{S2} EC [discPulAPulV](#)>

<x y r_i W_r N g₁ θ N_{wg} W₁ L_S g₂ W₂ L₂ EC [ringPulADS](#)>

<x y r_i W_r g₁ θ W₁ L_S g₂ W₂ L₂ EC [ringPulADSV](#)>

<x y r_i W_r N g₁ θ₁ N_{wg} W₁ L_{S1} g₂ θ₂ W₂ L_{S2} EC [ringPulAPul](#)>

<x y r_i W_r g₁ θ₁ W₁ L_{S1} g₂ θ₂ W₂ L_{S2} EC [ringPulAPulV](#)>

<x y r_d N g L_c N_{wg} W L_S EC [discPulleyLCA](#)>

<x y r_d g L_c W L_S EC [discPulleyLCAV](#)>

<x y r_i W_r N g L_c N_{wg} W L_S EC [ringPulleyLCA](#)>

<x y r_i W_r g L_c W L_S EC [ringPulleyLCAV](#)>

<x y r_d N g₁ L_c N_{wg} W₁ L_S g₂ W₂ L₂ EC [discPulLCADS](#)>

<x y r_d g₁ L_c W₁ L_S g₂ W₂ L₂ EC [discPulLCADSV](#)>

$\langle x \ y \ r_d \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ L_{c1} \ W_2 \ L_{S2} \ EC \ \text{discPulLCAPul} \rangle$

$\langle x \ y \ r_d \ \quad g_1 \ L_{c1} \quad W_1 \ L_{S1} \ g_2 \ L_{c1} \ W_2 \ L_{S2} \ EC \ \text{discPulLCAPulV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ \text{ringPulLCADS} \rangle$

$\langle x \ y \ r_i \ W_r \ \quad g_1 \ L_c \quad W_1 \ L_S \ g_2 \ W_2 \ L_2 \ EC \ \text{ringPulLCADSV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ L_{S1} \ g_2 \ L_{c1} \ W_2 \ L_{S2} \ EC \ \text{ringPulLCAPul} \rangle$

$\langle x \ y \ r_i \ W_r \ \quad g_1 \ L_{c1} \quad W_1 \ L_{S1} \ g_2 \ L_{c1} \ W_2 \ L_{S2} \ EC \ \text{ringPulLCAPulV} \rangle$

0.1.2.6 Disc-Ring Pulley Inverse Arc

<x y r_d N g θ N_{wg} W W_e L_S EC discPulleyInvA>

<x y r_d g θ W W_e L_S EC discPulleyInvAV>

<x y r_i W_r N g θ N_{wg} W W_e L_S EC ringPulleyInvA>

<x y r_i W_r g θ W W_e L_S EC ringPulleyInvAV>

<x y r_d N g₁ θ N_{wg} W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discPullInvADS>

<x y r_d g₁ θ W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discPullInvADSV>

<x y r_d N g₁ θ ₁ N_{wg} W₁ W_{e1} L_{S1} g₂ θ ₂ W₂ W_{e2} L_{S2} EC discPullInvAPul>

<x y r_d g₁ θ ₁ W₁ W_{e1} L_{S1} g₂ θ ₂ W₂ W_{e2} L_{S2} EC discPullInvAPulV>

<x y r_i W_r N g₁ θ N_{wg} W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC ringPullInvADS>

<x y r_i W_r g₁ θ W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC ringPullInvADSV>

<x y r_i W_r N g₁ θ ₁ N_{wg} W₁ W_{e1} L_S g₂ θ ₂ W₂ W_{e2} L₂ EC ringPullInvAPul>

<x y r_i W_r g₁ θ ₁ W₁ W_{e1} L_S g₂ θ ₂ W₂ W_{e2} L₂ EC ringPullInvAPulV>

<x y r_d N g L_c N_{wg} W W_e L_S EC discPulleyInvLCA>

<x y r_d g L_c W W_e L_S EC discPulleyInvLCAV>

<x y r_i W_r N g L_c N_{wg} W W_e L_S EC ringPulleyInvLCA>

<x y r_i W_r g L_c W W_e L_S EC ringPulleyInvLCAV>

<x y r_d N g₁ L_c N_{wg} W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discPullInvLCADS>

<x y r_d g₁ L_c W₁ W_{e1} L_S g₂ W₂ W_{e2} L₂ EC discPullInvLCADSV>

$\langle x \ y \ r_d \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{discPullInvLCAPul} \rangle$

$\langle x \ y \ r_d \ \quad g_1 \ L_{c1} \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{discPullInvLCAPulV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringPullInvLCADS} \rangle$

$\langle x \ y \ r_i \ W_r \ \quad g_1 \ L_c \quad W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringPullInvLCADSV} \rangle$

$\langle x \ y \ r_i \ W_r \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringPullInvLCADPul} \rangle$

$\langle x \ y \ r_i \ W_r \ \quad g_1 \ L_{c1} \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringPullInvLCADPulV} \rangle$

0.1.2.7 Disc-Ring Pulley Inverse Positive Tone Arc

<x y r_d r_e N g θ N_{wg} W W_e L_S EC discPulleyInvPosA>

<x y r_d r_e g θ W W_e L_S EC discPulleyInvPosAV>

<x y r_i W_r r_e N g θ N_{wg} W W_e L_S EC ringPulleyInvPosA>

<x y r_i W_r r_e g θ W W_e L_S EC ringPulleyInvPosAV>

<x y r_d r_e N g_1 θ N_{wg} W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC discPullInvPADS>

<x y r_d r_e g_1 θ W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC discPullInvPADSV>

<x y r_d r_e N g_1 θ_1 N_{wg} W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_{S2} EC discPullInvPAPul>

<x y r_d r_e g_1 θ_1 W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_{S2} EC discPullInvPAPulV>

<x y r_i W_r r_e N g_1 θ N_{wg} W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC ringPullInvPADS>

<x y r_i W_r r_e g_1 θ W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC ringPullInvPADSV>

<x y r_i W_r r_e N g_1 θ_1 N_{wg} W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_2 EC ringPullInvPAPul>

<x y r_i W_r r_e g_1 θ_1 W_1 W_{e1} L_{S1} g_2 θ_2 W_2 W_{e2} L_2 EC ringPullInvPAPulV>

<x y r_d r_e N g L_c N_{wg} W W_e L_S EC discPulleyInvPosLCA>

<x y r_d r_e g L_c W W_e L_S EC discPulleyInvPosLCAV>

<x y r_i W_r r_e N g L_c N_{wg} W W_e L_S EC ringPulleyInvPosLCA>

<x y r_i W_r r_e g L_c W W_e L_S EC ringPulleyInvPosLCAV>

<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>

<x y r_d r_e g_1 L_c W_1 W_{e1} L_S g_2 W_2 W_{e2} L_2 EC discPullInvPLCADSV>

$\langle x \ y \ r_d \ r_e \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_2 \ EC \ \text{discPullInvPLCAPul} \rangle$

$\langle x \ y \ r_d \ r_e \ \quad g_1 \ L_{c1} \quad \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_2 \ EC \ \text{discPullInvPLCAPulV} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_c \ N_{wg} \ W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringPullInvPLCADS} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ \quad g_1 \ L_c \quad \quad W_1 \ W_{e1} \ L_S \ g_2 \ W_2 \ W_{e2} \ L_2 \ EC \ \text{ringPullInvPLCADSV} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ N \ g_1 \ L_{c1} \ N_{wg} \ W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringPullInvPLCAPul} \rangle$

$\langle x \ y \ r_i \ W_r \ r_e \ \quad g_1 \ L_{c1} \quad \quad W_1 \ W_{e1} \ L_{S1} \ g_2 \ L_{c2} \ W_2 \ W_{e2} \ L_{S2} \ EC \ \text{ringPullInvPLCAPulV} \rangle$

0.1.3 Waveguides

0.1.3.1 180 degree bend with taper

180 degree bend of diameter D connects two waveguides of length L_1 and L_2 . The 180 degree connection has a linearly varying width from W to W_t . Integer variable N represents the number of vertices that construct the 180 degree bend.

< x y L_1 L_2 D W W_t N $\theta_{x,y}$ **180degreeBendT**>

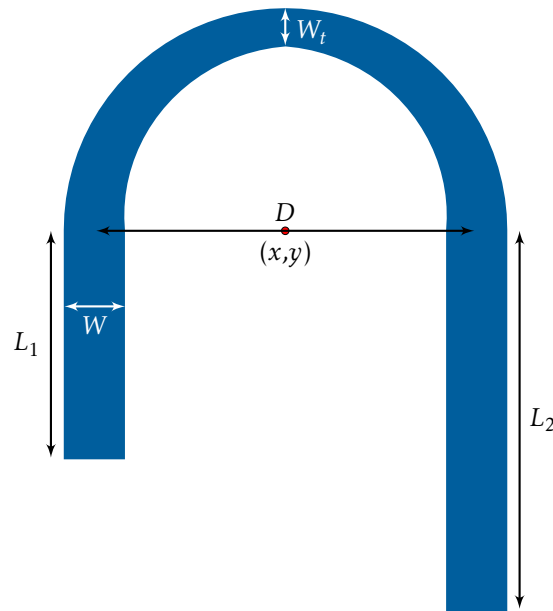


Figure 8: Example of a **180degreeBendT** rendered GDS output.

0.1.3.2 Racetrack-Waveguides

The structure in figure 9 shows a racetrack with waveguides above and below. To eliminate either of the waveguide structures, the respective length (L_1 or L_2) is set to zero. The parameter EC controls if semicircular endcaps are incorporated in the waveguide structure ($EC = 0$ and $EC = 1$ for waveguides without and with endcaps, respectively).

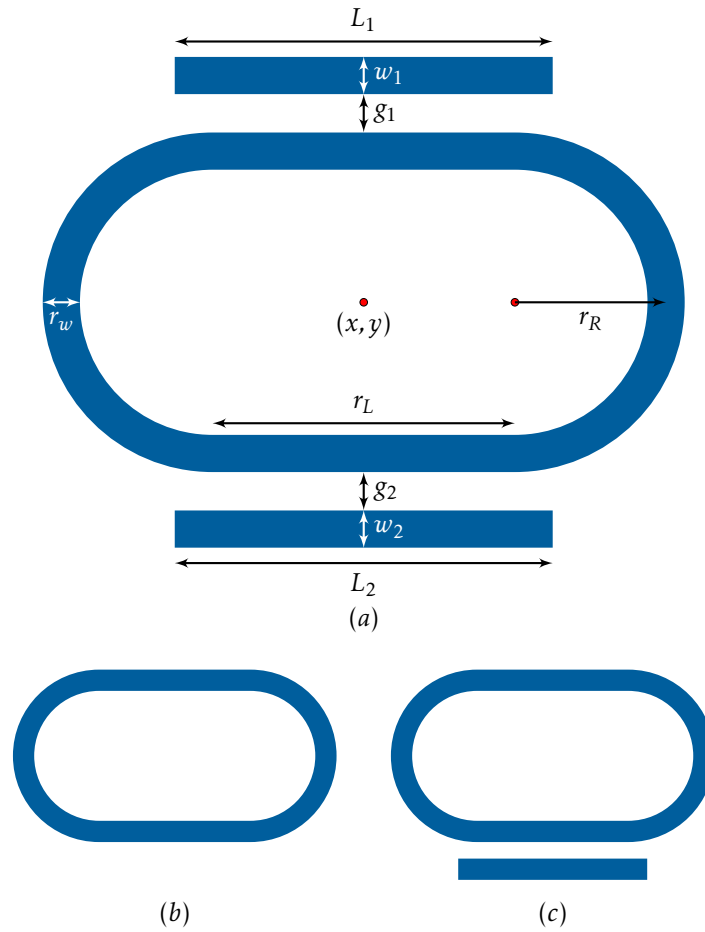


Figure 9: Racetrack structure with (a) waveguides above and below, (b) without waveguides ($L_1 = L_2 = 0$) and (c) with a single waveguide below ($L_1 = 0$).

$\langle x \ y \ r_L \ r_w \ r_R \ N_R \ L_1 \ g_1 \ w_1 \ L_2 \ g_2 \ w_2 \ \theta_{(x,y)} \ EC \ \text{raceTrackWG} \rangle$

0.1.3.3 Racetrack-Waveguides Inverse

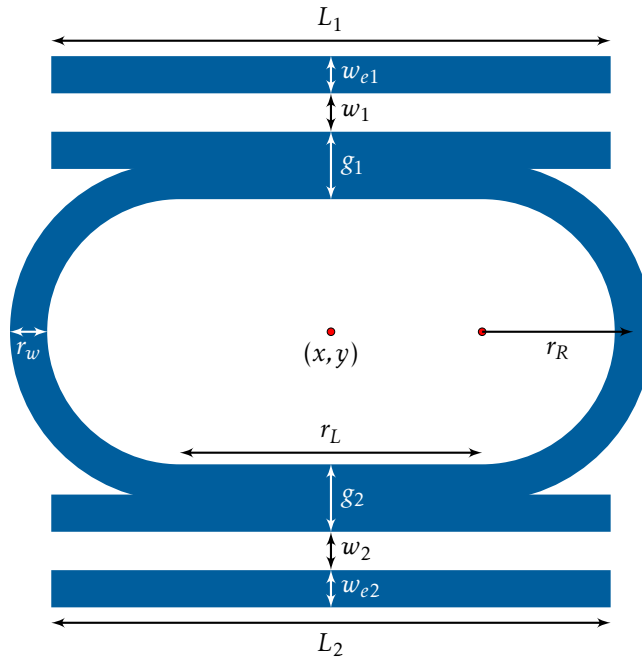


Figure 10: Example illustrating various parameters of the `raceTrackWGi` constructor.

`<x y r_L r_w r_R N_R L_1 g_1 w_1 w_{e1} L_2 g_2 w_2 w_{e2} \theta_{(x,y)} EC raceTrackWGi>`

0.1.3.4 Racetrack-Waveguides Positive Inverse

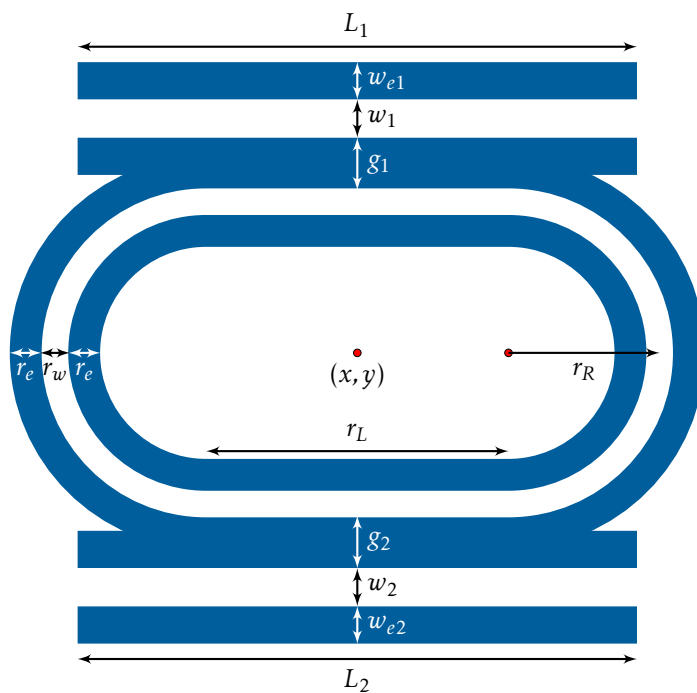


Figure 11: Example illustrating various parameters of the `raceTrackWGpi` constructor.

`<x y r_L r_w r_e r_R N_R L_1 g_1 w_1 w_{e1} L_2 g_2 w_2 w_{e2} \theta_{(x,y)} EC raceTrackWGpi>`

0.1.3.5 Racetrack-Waveguides Symmetric

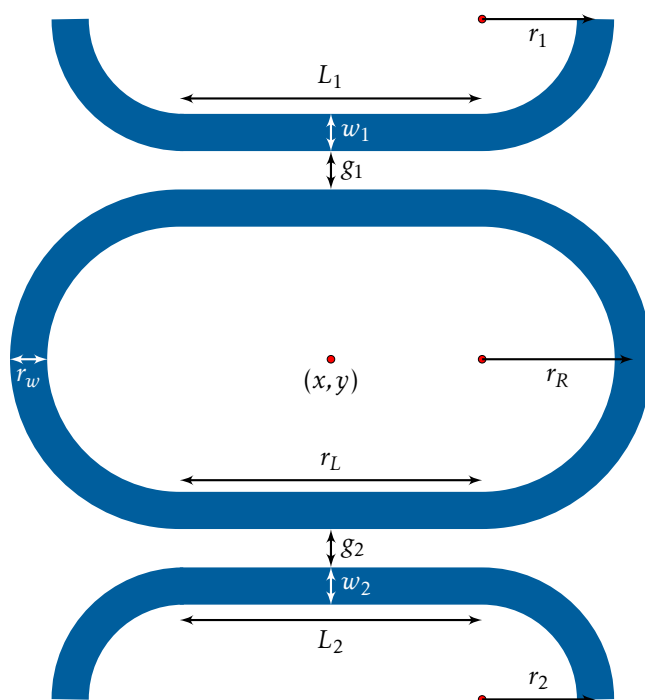


Figure 12: Example illustrating various parameters of the `raceTrackWGs` constructor.

`<x y r_L r_w r_R N_R L_1 g_1 w_1 r_1 N_1 L_2 g_2 w_2 r_2 N_2 \theta_{(x,y)} EC raceTrackWGs>`

0.1.3.6 Racetrack-Waveguides Symmetric Inverse

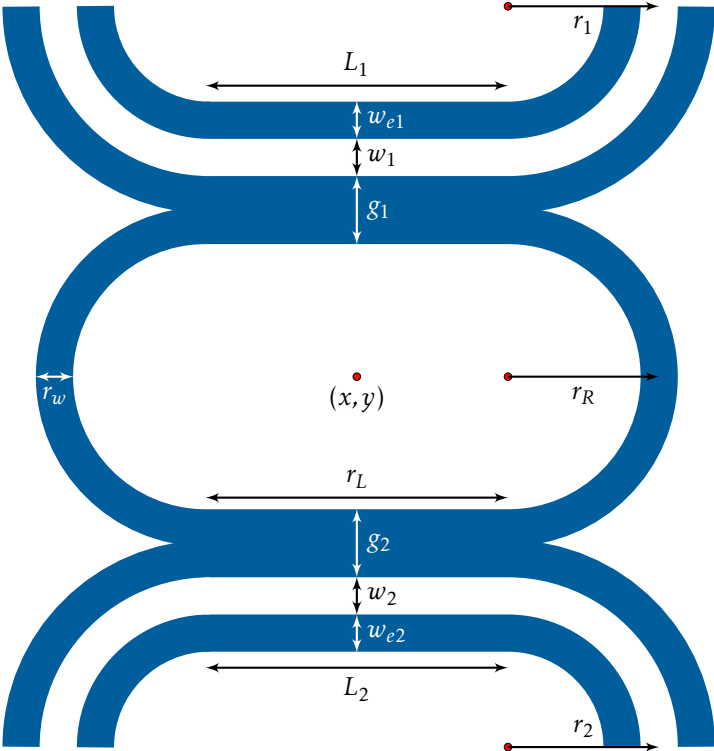


Figure 13: Example illustrating various parameters of the `raceTrackWGsi` constructor.

 $\langle x \ y \ r_L \ r_w \ r_R \ N_R \ L_1 \ g_1 \ w_1 \ w_{e1} \ r_1 \ N_1 \ L_2 \ g_2 \ w_2 \ w_{e2} \ r_2 \ N_2 \ \theta_{(x,y)} \ EC \ \text{raceTrackWGsi} \rangle$

0.1.3.7 Racetrack-Waveguides Symmetric Positive Inverse

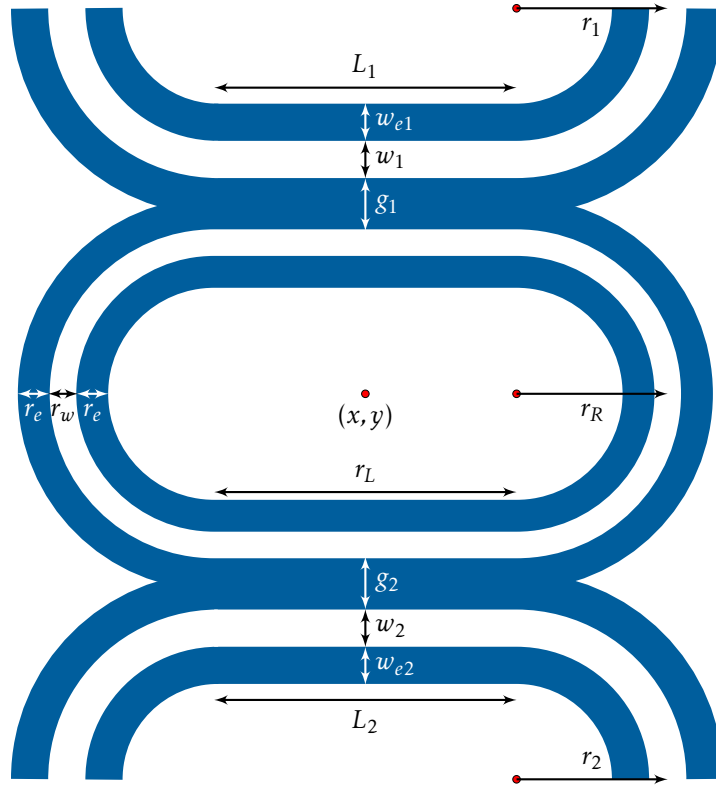


Figure 14: Example illustrating various parameters of the `raceTrackWGspi` constructor.

`<x y r_L r_w r_e r_R N_R L_1 g_1 w_1 w_{e1} r_1 N_1 L_2 g_2 w_2 w_{e2} r_2 N_2 \theta_{(x,y)} EC raceTrackWGspi>`

0.1.3.8 Waveguide with linear taper

Waveguide of width W , length L , taper length L_t and taper width W_t .

$\langle x \ y \ L \ W \ L_t \ W_t \ \theta_{x,y} \ \text{wgLinearTaper} \rangle$

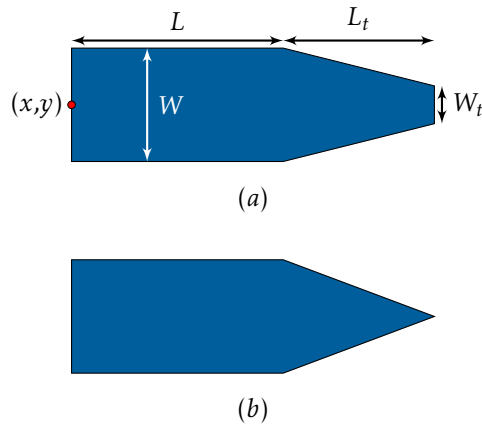


Figure 15: (a) Example of a *wgLinearTaper*. (b) Waveguide Linear taper structure with $W_t = 0$.

0.1.3.9 Waveguide Tapers With Supports

Constructors **taperSB** and **taperSBinv** create a tapered waveguide with rounded supports. The former creates two shapes, a rounded rectangular box in GDS layer b_{Lyr} and the waveguide taper with supports in the user defined GDS layer. The **taperSBinv** version creates an inverse waveguide tapered structure with supports in a user defined GDS layer. These structures can contain many supporting elements. Corner rounding is vectorized and defined by **shapeReso**.

The **taperSupport** structure can be used in conjunction with **taperSB**. Similar to Lego building blocks, users can concatenate numerous tapered and straight segments in an area defined by a width b_W and height b_H . Depending on the tone of the defining resist, the final structure may need to be inverted.

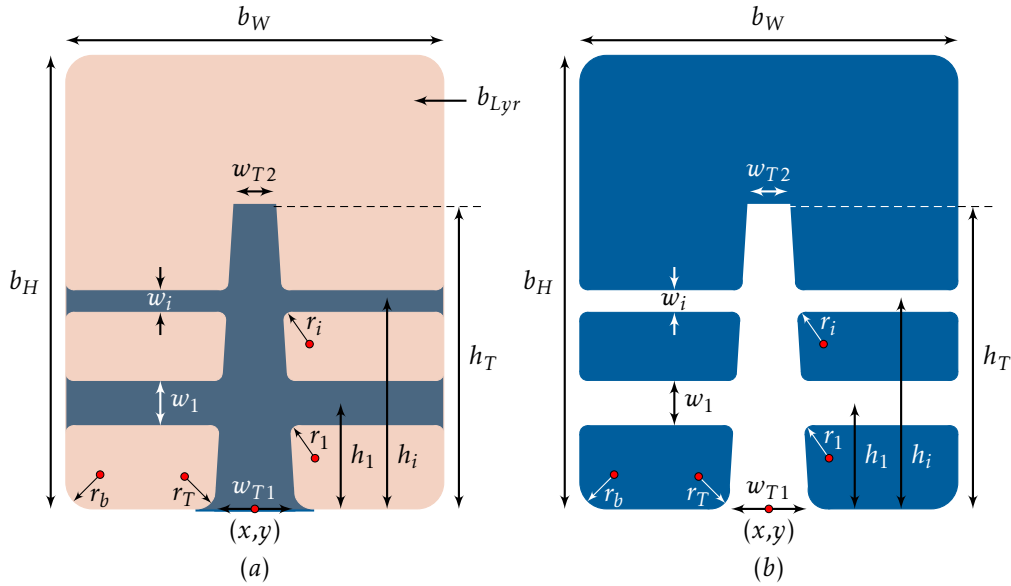


Figure 16: Waveguide tapers with rounded, stress relief supports. a) **taperSB** structure containing the round rectangle in GDS layer b_{Lyr} and a waveguide taper with supports. (b) **taperSBinv** inverse structure cast into a user defined GDS layer.

x y b_W b_H r_b b_{Lyr} w_{T1} w_{T2} h_T r_T H_1 w_1 r_1 ... H_i w_i r_i $\theta_{(x,y)}$ **taperSB**

x y b_W b_H r_b w_{T1} w_{T2} h_T r_T H_1 w_1 r_1 ... H_i w_i r_i $\theta_{(x,y)}$ **taperSBinv**

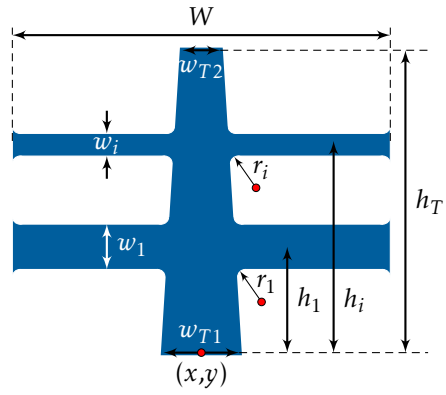


Figure 17: Waveguide tapers with rounded, stress relief supports created using the **taperSupport** constructor.

x y W w_{T1} w_{T2} h_T H_1 w_1 r_1 ... H_i w_i r_i $\theta_{(x,y)}$ **taperSupport**

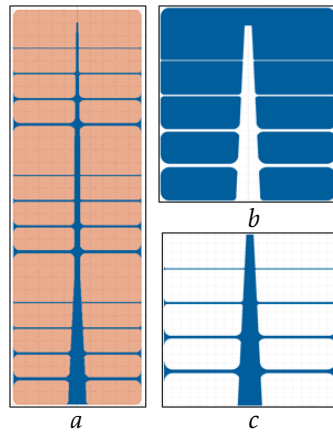


Figure 18: Rendered GDS shapes for the (a) **taperSB** with various **taperSupport** structures, (b) **taperSBinv** and (c) **taperSupport** constructors.

0.1.3.10 Waveguide Wave

Constructor `waveguideWave` creates an input and an output rectangular waveguide connected by a bezier expander. The waveguides are of width W and lengths L_1 and L_2 . The s-Bend bezier connection is of amplitude A and length L .

`<x y L1 L2 W L A $\theta_{(x,y)}$ waveguideWave>`

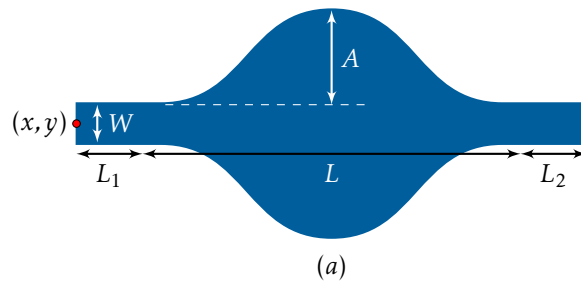


Figure 19: Example of a `waveguideWave` structure.

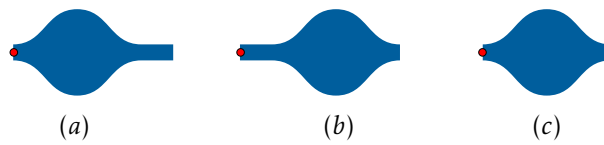


Figure 20: Examples of a `waveguideWave` structure with (a) $L_1 = 0$, (b) $L_2 = 0$ and (c) $L_1 = L_2 = 0$.

0.1.3.11 Waveguide Wave Array

Constructor `waveguideWaveArray` creates an array of N `waveGuideWave` elements. A GDS structure named `uniqueStructName` is created with a single `waveguideWave` element that is then instantiated within the active GDS struct. The waveguides are of width W and lengths L_1 and L_2 . The s-Bend bezier connection is of amplitude A and length L . Figure 21(b) shows an array with input and output rectangular waveguide elements of length L_1 and L_2 . Figure 21 (c) shows an array of N elements with $L_1 = L_2 = 0$.

`<uniqueStructName x y L1 L2 W L A N waveguideWaveArray>`

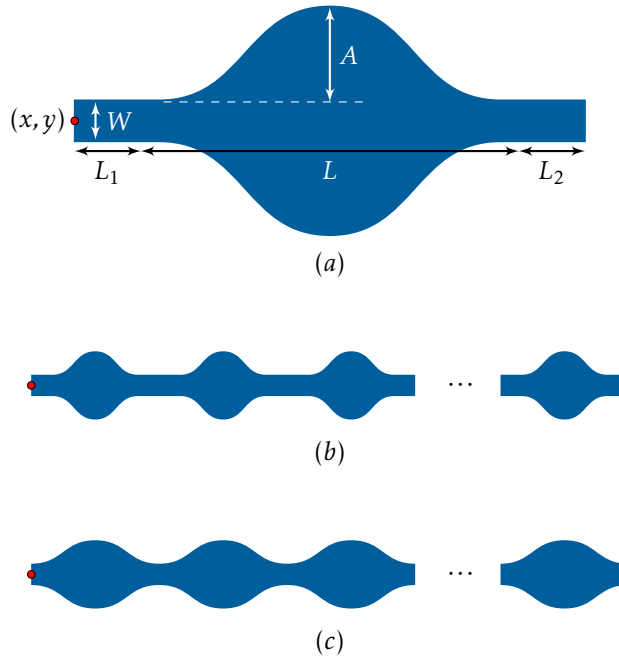


Figure 21: Example of a `waveguideWaveArray` constructor. (a) Single `waveguideWaveArray` element, array of N -repetitions with (b) rectangular input-output ports and (c) $L_1 = L_2 = 0$.

0.2 Arrays and Instances

0.2.1 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 Δx and Δy 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.

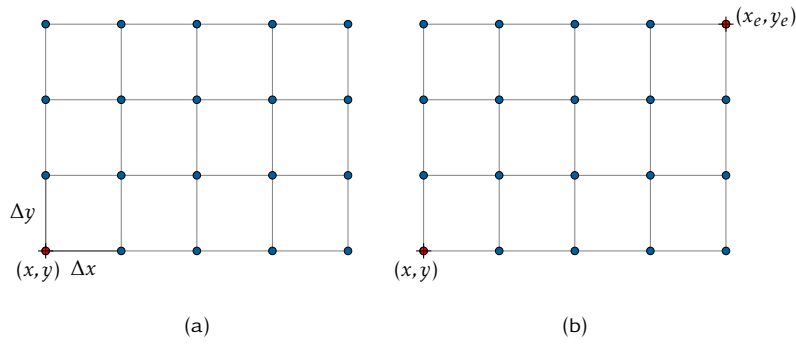


Figure 22: Two versions of the rectangular array constructor `arrayRect` (a) 1 (b) 2.

```
<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>
```

0.2.2 Rectangular Array - Centered

The `arrayRectC` constructor centers a rectangular array around the coordinate (x_c, y_c) . Parameters Δx and Δy are the pitch between the columns and rows, respectively.

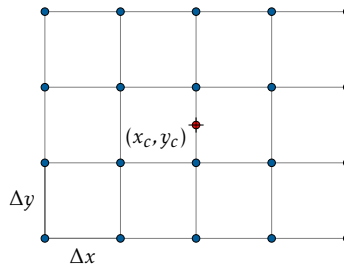


Figure 23: Centered rectangular array constructor `arrayRectC`.

`<structureToBeArrayed xc yc Ncolumns Nrows Δx Δy arrayRectC>`

0.2.3 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 ($N_{columns}$ and N_{rows}) are specified, with previously defined structure instantiated on a hexagonal grid with spacing defined by Δs .

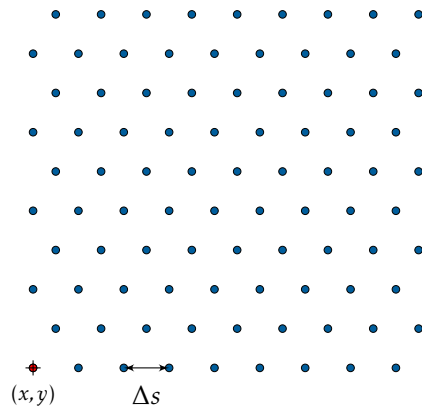


Figure 24: `arrayHex` constructor used to create hexagonal arrays.

`<structureToBeArrayed xc yc Ncolumns Nrows Δs arrayHex>`

0.2.4 Hexagonal Array - Centered

The `arrayHexC` constructor centers a hexagonal array around the coordinate (x_c, y_c) .

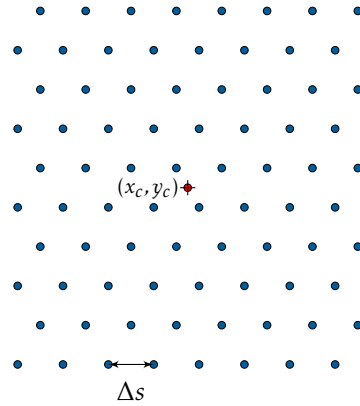


Figure 25: `arrayHexC` constructor used to create centered hexagonal arrays.

`<structureToBeArrayed x_c y_c N_columns N_rows Δs arrayHexC>`

0.3 Text Labels, PostScript and Logos

0.3.1 Text

Update 2016 release of the Nanolithography Toolbox manual with this page. 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 26: GDS rendered `textgds` example.

The following are scripts used to generate the text in Figure 26:

```
<{{SomeText!@#}} {{Algerian}} 20 0 0 textgds>
```

```
<{{4!AaBbCc}} {{Serif}} 20 0 0 textgds>
```

```
<{{4!AaBbCc}} {{Arial}} 20 0 0 textgds>
```

0.3.2 Text - Outline

Update 2016 release of the Nanolithography Toolbox manual with this page. 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 *font-Name* 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 0.9.5).

```
<{{someText}} {{fontName}} fontSize x y textOutline>
```

```
<{{someText}} {{fontName}} fontSize x y textOutlineC>
```



Figure 27: Text Outline GDS example.

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.

0.3.3 Text - Fill

During structural release, inner segments of objects could be released and potentially contaminate device samples. Text fill constructor circumvents these dilemmas by creating solid letters without inner boundaries. The position (x, y) of the text fill object is set by either the lower left coordinate (`textFill`) or the center (`textFillC`).

```
<{{someText}} {{fontName}} fontSize x y textFill>  
<{{someText}} {{fontName}} fontSize x y textFillC>
```

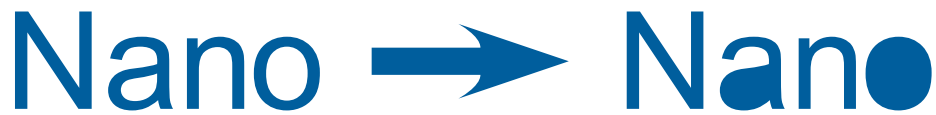


Figure 28: Juxtaposed `textgds` (left) and the `textFill` (right) rendered GDS outputs. The inner boundaries are removed within the `textFill` objects.

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.

0.3.4 Text - Outline Fill

During structural release, inner segments of objects could be released and potentially contaminate device samples. Text outline fill constructor circumvents these dilemmas by creating outlines of letters without inner boundaries. The position (x,y) of the text outline fill object is set by either the lower left coordinate (`textOutlineFill`) or the center (`textOutlineFillC`).

```
<{{someText}} {{fontName}} fontSize x y textOutlineFill>  
<{{someText}} {{fontName}} fontSize x y textOutlineFillC>
```

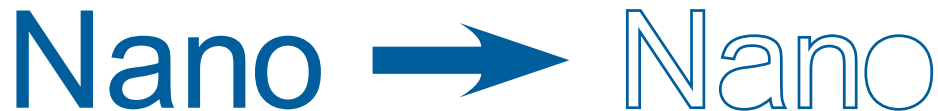


Figure 29: Juxtaposed `textgds` (left) and the `textOutlineFill` (right) rendered GDS outputs. The inner boundaries are removed within the `textFill` objects.

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.

0.3.5 Text - Dashed

During structural release, inner segments of objects could be released and potentially contaminate device samples. Text dashed constructor circumvents these dilemmas by creating dashed lines of letters without continuous boundaries. The position (x, y) of the text dashed object is set by either the lower left coordinate (`textDashed`) or the center (`textDashedC`). Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

```
<{{someText}} {{fontName}} fontSize x y d_L d_G Cap Join textDashed>  
<{{someText}} {{fontName}} fontSize x y d_L d_G Cap Join textDashed>
```

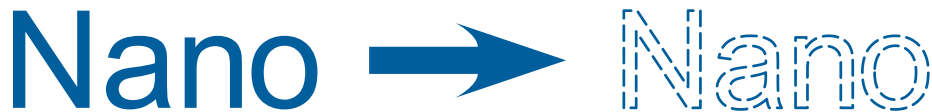


Figure 30: Example of a `textDashed` rendered GDS output.

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.

0.3.6 Label Maker

Update 2016 release of the Nanolithography Toolbox manual with this page. 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* in μm), and the pitch along the two directions (Δ_x and Δ_y) are specified. Similarly, custom labels are defined by an additional set of label parameters ($L_1, L_2 \dots L_n$). In all cases, if the font name is not contained within the system font list, label maker will default to a Serif font. Constructor parameters shown below must be TAB separated.

4 auto labels:

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y autoOut labelMaker}
```

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y autoRowCol labelMaker}
```

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y autoOutLett labelMaker}
```

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y autoRowColLett labelMaker}
```

2 custom labels:

```
{L_1 L_2 ... L_n N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y out labelMaker}
```

```
{L_1 L_2 ... L_n N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y rowCol labelMaker}
```

IMPORTANT NOTES: 1) Labels $L_1 \dots L_n$ 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.

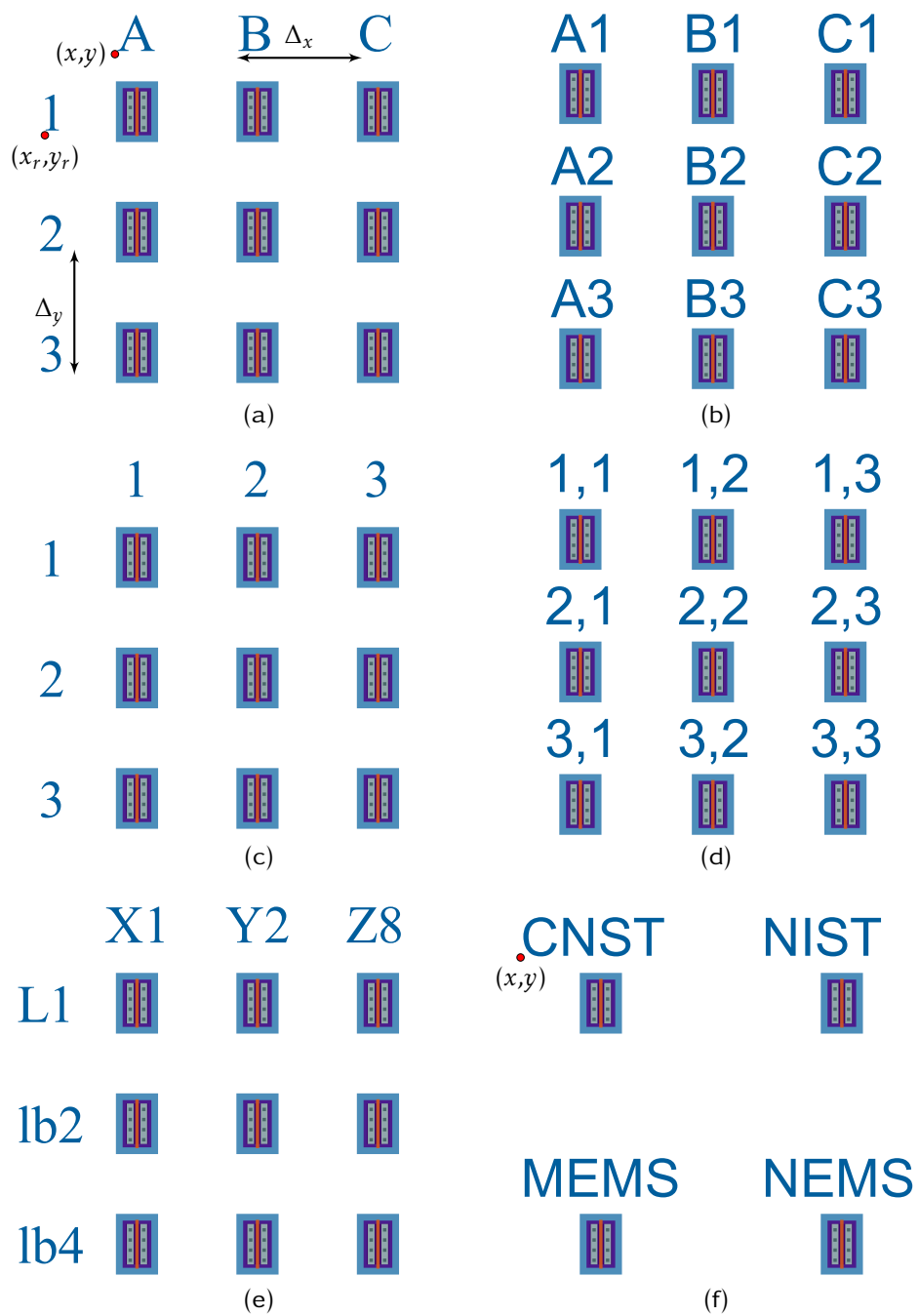


Figure 31: Label Maker Constructor Examples: (a) *autoOutLett*, (b) *autoRowColLett*, (c) *autoOut*, (d) *autoRowCol*, (e) *out*, and (f) *rowCol*.

0.3.7 Label Maker - Outline Text

Update 2016 release of the Nanolithography Toolbox manual with this page. This method is identical to the previously defined label maker section (0.3.6) with the exception that rendered shapes are outlined text objects. Outline width is specified using the **fontOutline** parameter (see section 0.9.5). Constructor parameters shown below must be **TAB** separated.

4 auto labels:

$\{N_{row} N_{col} \text{ fontName } \text{fontSize } x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOut} \quad \text{labelOutline}\}$

$\{N_{row} N_{col} \text{ fontName } \text{fontSize } x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowCol} \quad \text{labelOutline}\}$

$\{N_{row} N_{col} \text{ fontName } \text{fontSize } x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOutLett} \quad \text{labelOutline}\}$

$\{N_{row} N_{col} \text{ fontName } \text{fontSize } x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowColLett} \quad \text{labelOutline}\}$

2 custom labels:

$\{L_1 \ L_2 \dots L_n \ N_{row} \ N_{col} \ \text{fontName } \text{fontSize } x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{out} \quad \text{labelOutline}\}$

$\{L_1 \ L_2 \dots L_n \ N_{row} \ N_{col} \ \text{fontName } \text{fontSize } x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{rowCol} \quad \text{labelOutline}\}$

IMPORTANT NOTES: 1) Labels $L_1 \dots L_n$ 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.

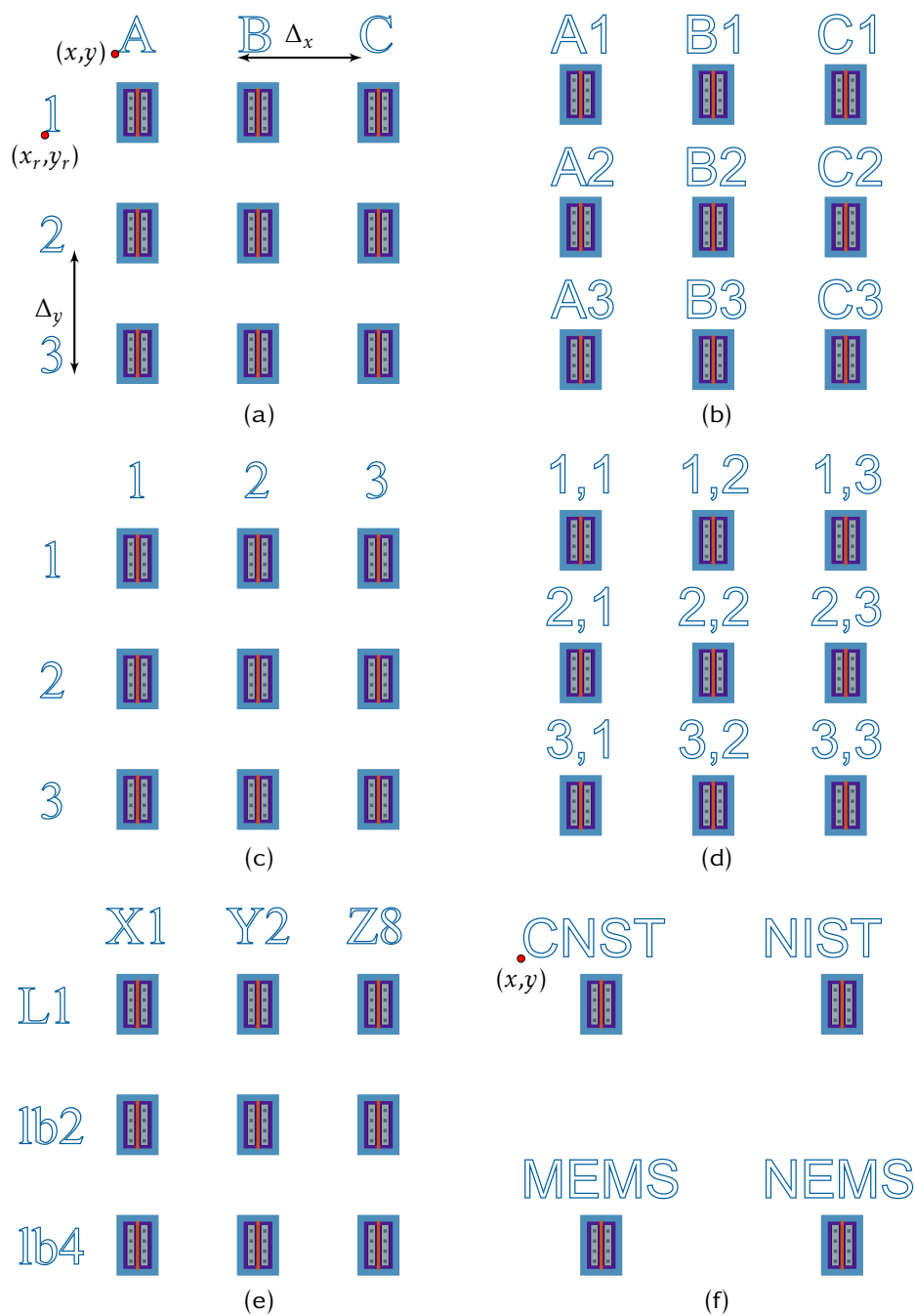


Figure 32: Label maker outline constructor examples: (a) *autoOutLett*, (b) *autoRowColLett*, (c) *autoOut*, (d) *autoRowCol*, (e) *out*, and (f) *rowCol*.

0.3.8 Label Maker - Fill Text

Below are constructors for creating labels using the text fill objects. These are similar to [labelMaker](#) and [labelOutline](#) constructors. Detailed description is included with the [labelMaker](#) constructor. Constructor parameters shown below must be TAB separated.

4 auto labels:

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOut} \ \text{labelFill}\}$

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowCol} \ \text{labelFill}\}$

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOutLett} \ \text{labelFill}\}$

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowColLett} \ \text{labelFill}\}$

2 custom labels:

$\{L_1 \ L_2 \dots L_n \ N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{out} \ \text{labelFill}\}$

$\{L_1 \ L_2 \dots L_n \ N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{rowCol} \ \text{labelFill}\}$

IMPORTANT NOTES: 1) Labels $L_1 \dots L_n$ 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.

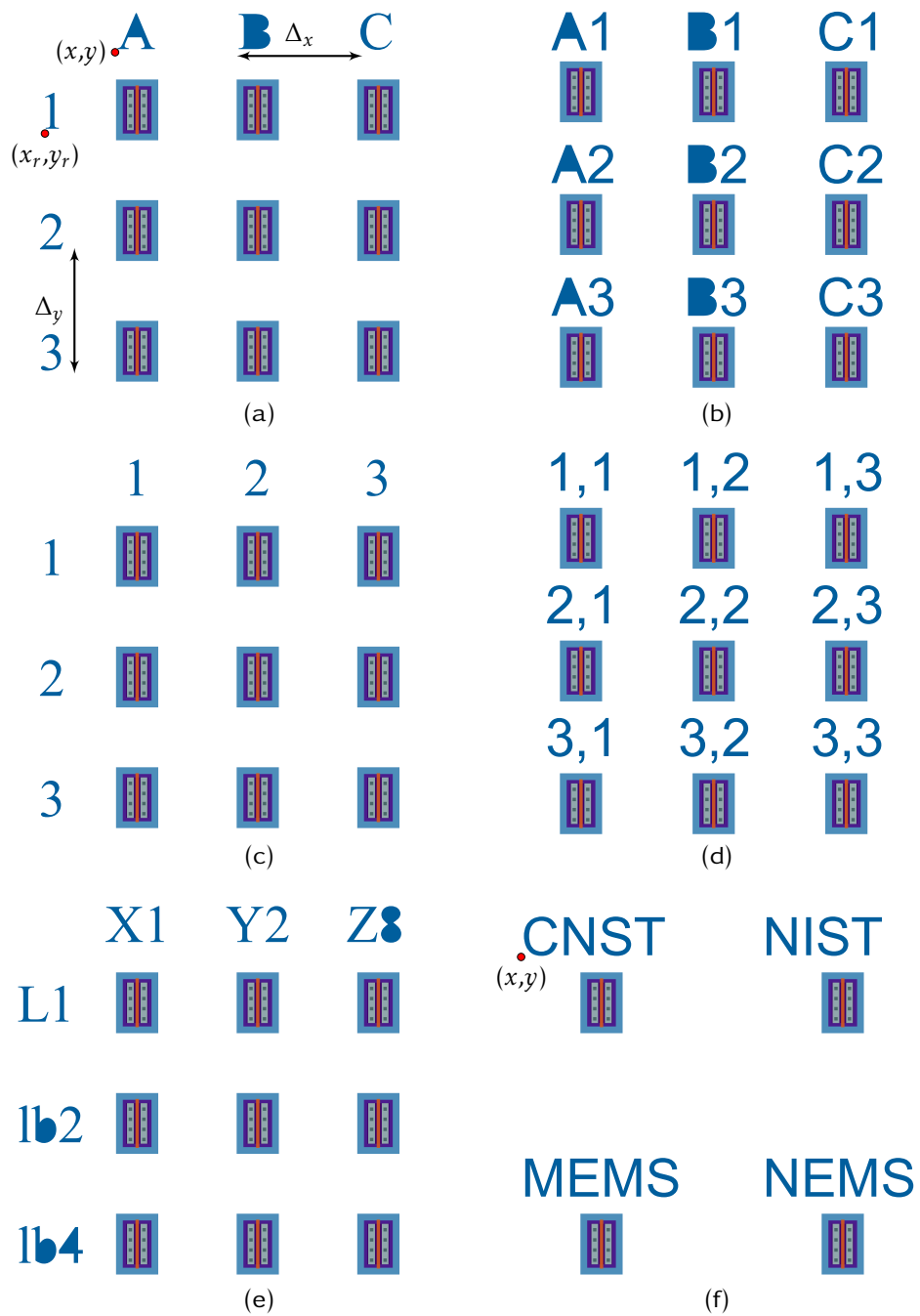


Figure 33: Label maker fill constructor examples: (a) *autoOutLett*, (b) *autoRowColLett*, (c) *autoOut*, (d) *autoRowCol*, (e) *out*, and (f) *rowCol*.

0.3.9 Label Maker - Outline Fill Text

Below are constructors for creating labels using the text outline fill objects. These are similar to **labelMaker** and **labelOutline** constructors. Detailed description is included with the **labelMaker** constructor. Constructor parameters shown below must be **TAB** separated.

4 auto labels:

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOut} \ \text{labelOutlineFill}\}$

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowCol} \ \text{labelOutlineFill}\}$

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoOutLett} \ \text{labelOutlineFill}\}$

$\{N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{autoRowColLett} \ \text{labelOutlineFill}\}$

2 custom labels:

$\{L_1 \ L_2 \dots L_n \ N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{out} \ \text{labelOutlineFill}\}$

$\{L_1 \ L_2 \dots L_n \ N_{row} \ N_{col} \ fontName \ fontSize \ x \ y \ x_r \ y_r \ \Delta_x \ \Delta_y \ \text{rowCol} \ \text{labelOutlineFill}\}$

IMPORTANT NOTES: 1) Labels $L_1 \dots L_n$ 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.

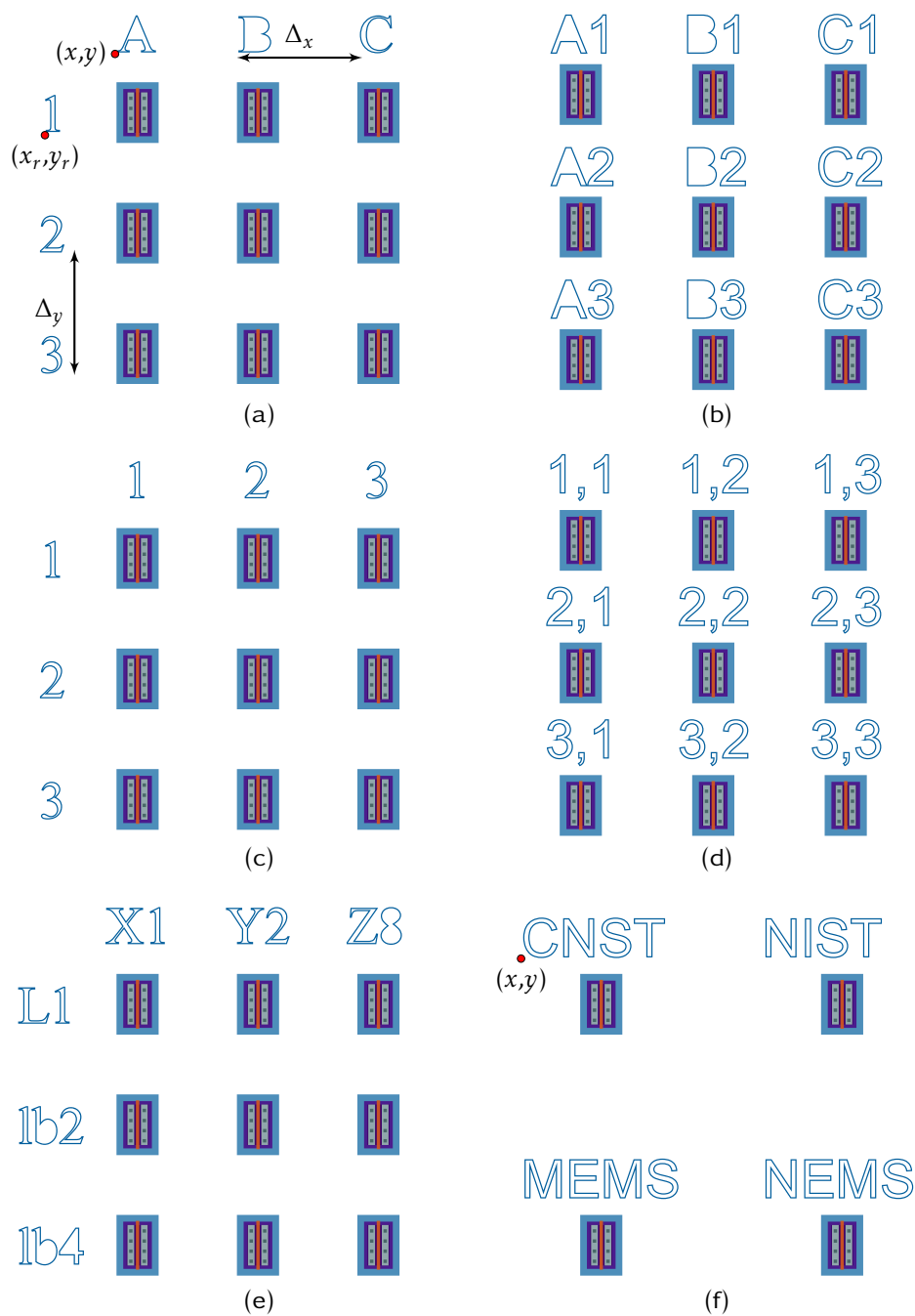


Figure 34: Label Maker Outline Fill constructor examples: (a) *autoOutLett*, (b) *autoRowColLett*, (c) *autoOut*, (d) *autoRowCol*, (e) *out*, and (f) *rowCol*.

0.3.10 Label Maker - Dashed Text

Below are constructors for creating labels using the text outline fill objects. These are similar to **labelMaker** and **labelOutline** constructors. Detailed description is included with the **labelMaker** constructor. Dash length and gap are defined by d_L and d_G , respectively. The C and J, shown in Figure 116, define the end-cap and line-join parameters. Constructor parameters shown below must be **TAB** separated.

4 auto labels:

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y d_L d_G C J autoOut labelDashed}
```

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y d_L d_G C J autoRowCol labelDashed}
```

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y d_L d_G C J autoOutLett labelDashed}
```

```
{N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y d_L d_G C J autoRowColLett labelDashed}
```

2 custom labels:

```
{L_1 L_2 ... L_n N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y d_L d_G C J out labelDashed}
```

```
{L_1 L_2 ... L_n N_row N_col fontName fontSize x y x_r y_r Δ_x Δ_y d_L d_G C J rowCol labelDashed}
```

IMPORTANT NOTES: 1) Labels $L_1 \dots L_n$ 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.

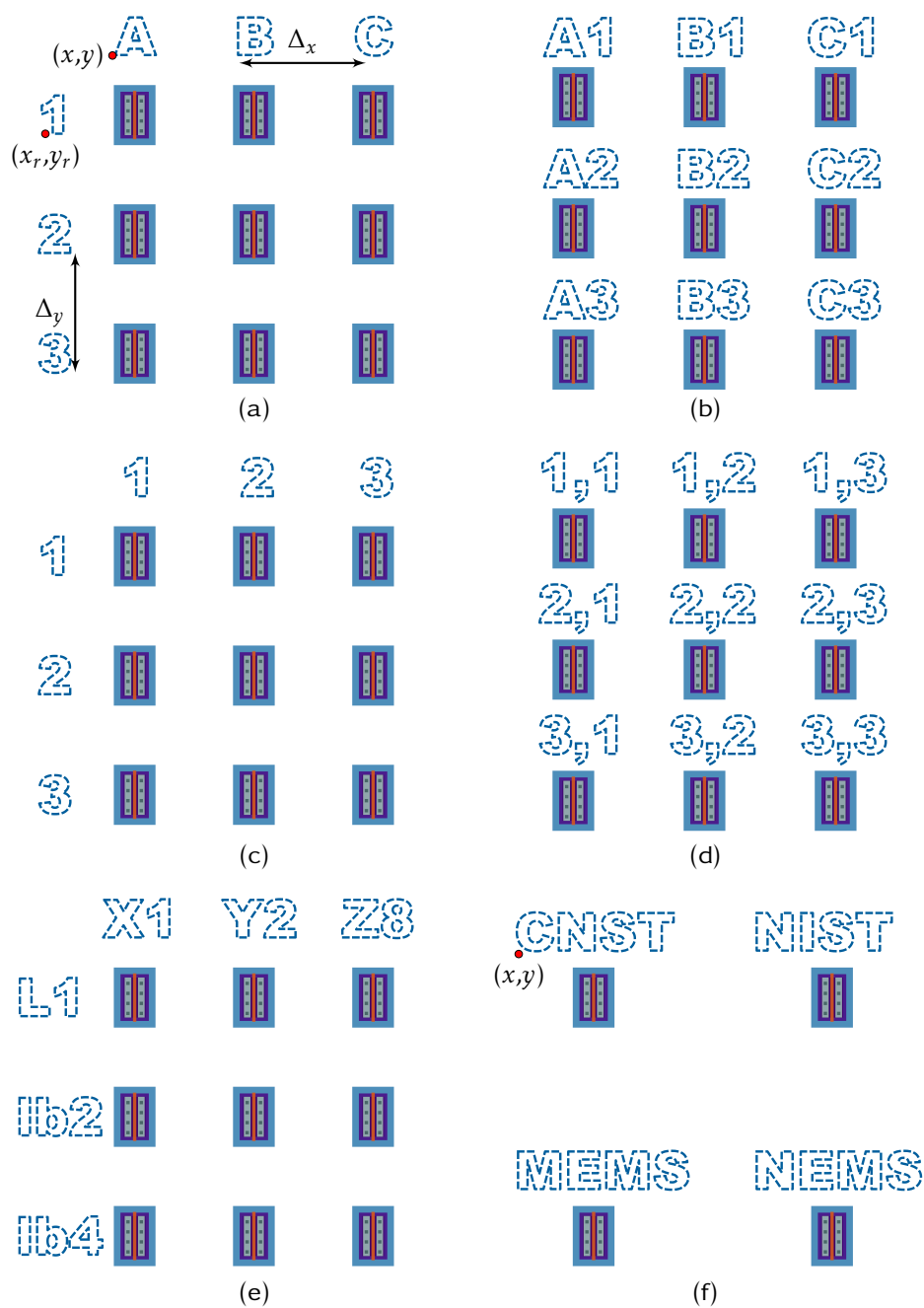


Figure 35: Label maker dashed constructor examples: (a) *autoOutLett*, (b) *autoRowColLett*, (c) *autoOut*, (d) *autoRowCol*, (e) *out*, and (f) *rowCol*.

0.3.11 CNST and NIST logos

Update 2016 release of the Nanolithography Toolbox manual with this page. 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.

`x` `y` `scale` `cnstEmblem`

`x` `y` `scale` `cnstLogo`

`x` `y` `scale` `nistLogo`

`x` `y` `scale` `nistCnstLogo`

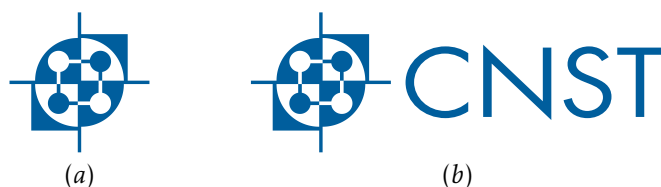


Figure 36: CNST logos created using the (a) `cnstLogoEmblem` and (b) `cnstLogo` constructors.



Figure 37: NIST logo created using the `nistLogo` constructor.



Figure 38: Combined NIST and CNST logos created using the `nistCnstLogo` constructor.

0.3.12 CNST and NIST logos - Outline and Dashed

The following constructors are outlined and dashed versions of the CNST and NIST logos. Figure 39 shows the [nistLogoOutline](#) and [nistLogoDashed](#) examples. The outline constructors are defined by scaling (see previous section 0.3.11) and by the shape outline width w_o . Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join parameters define the end-cap and line-join parameters.

Outline CNST and NIST logos:

x	y	$scale$	w_o	cnstEmblemOutline
x	y	$scale$	w_o	cnstLogoOutline
x	y	$scale$	w_o	nistLogoOutline
x	y	$scale$	w_o	nistCnstLogoOutline

Dashed CNST and NIST logos:

x	y	$scale$	w_o	d_L	d_G	Cap	$Join$	cnstEmblemDashed
x	y	$scale$	w_o	d_L	d_G	Cap	$Join$	cnstLogoDashed
x	y	$scale$	w_o	d_L	d_G	Cap	$Join$	nistLogoDashed
x	y	$scale$	w_o	d_L	d_G	Cap	$Join$	nistCnstLogoDashed

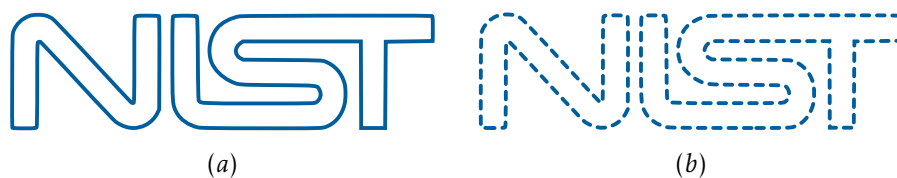


Figure 39: CNST logos created using the (a) [nistLogoOutline](#) and (b) [nistLogoDashed](#) constructors.

0.4 Objects - CNST Special Scripts

0.4.1 Archimedes spiral with skipped turns

The `spiralArchST` constructor defines an Archimedes spiral centered around (x, y) , of line width W , number of turns N_{turns} , separation between turns S , points per turn P_T , skipped turns S_T , end length L and rotation about the center coordinate $\theta(x, y)$.

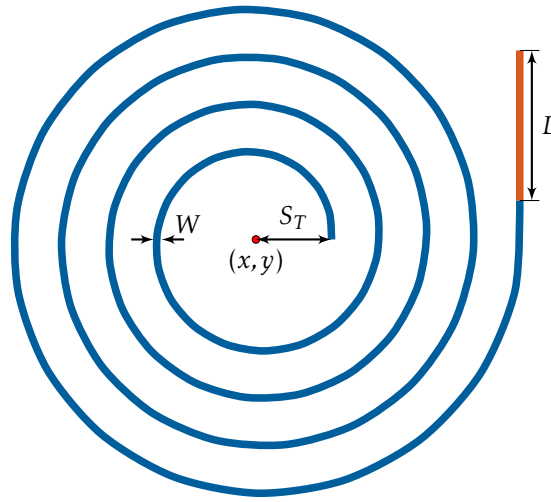


Figure 40: Archimedes spiral with skipped turns created using the `spiralArchST` constructor. Parameter S_T is an integer ($S_T \geq 0$) defines the number of skipped turns. $EC = 1$ or 0 corresponding to the waveguide coupling segment without and with semi-circular endcaps.

x y W N_{turns} S P_T S_T L $\theta_{(x,y)}$ EC `spiralArchST`

0.4.2 Crossing wires

N element cross-wire structure made from three GDS layers (L_a, L_b, L_c).

x y w_1 w_2 p_1 p_2 L_1 L_2 L_3 a b N L_a L_b L_c **crossWiresV1**

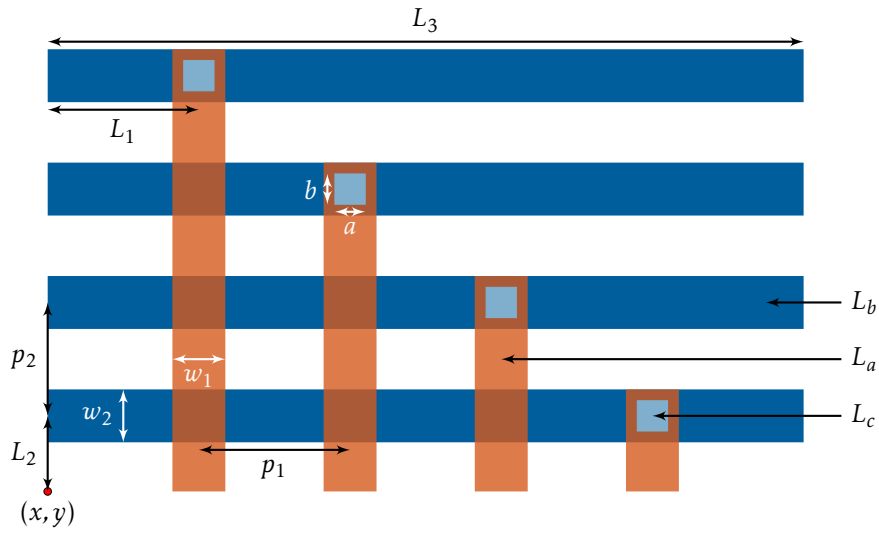


Figure 41: Cross-wire structure **crossWiresV1** with $N = 4$.

x y w_1 w_2 p_1 p_2 L_1 L_2 L_3 a b N L_a L_b L_c **crossWiresV2**

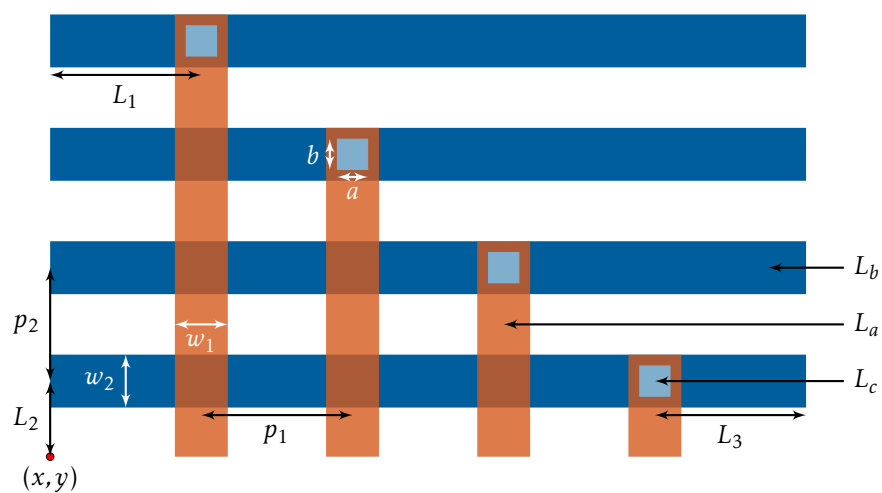


Figure 42: Cross-wire structure **crossWiresV2** with $N = 4$.

crossWiresV3

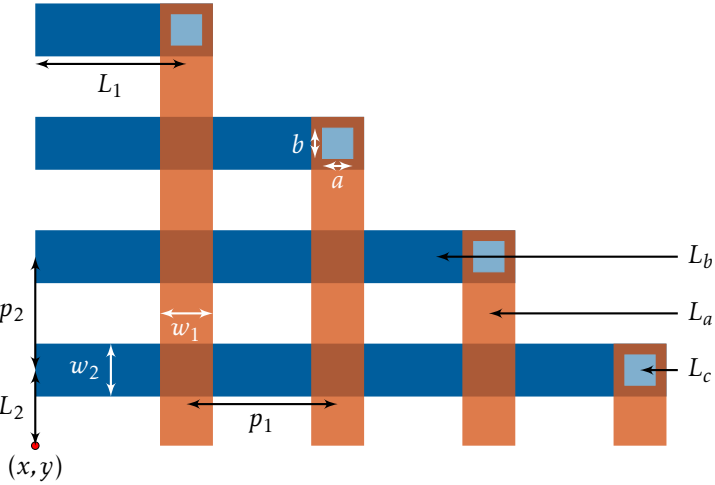


Figure 43: Cross-wire structure `crossWiresV3` with $N = 4$.

0.4.3 Elliptical Torus-Arc

Elliptical torus-arc structures defined by the origin and the focus coordinates.

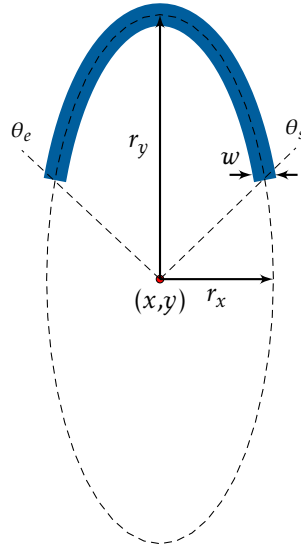


Figure 44: Elliptical torus-arc structure with the center defined by (x, y) .

x y r_x r_y w θ_s θ_e N_{sides} $\theta_{(x,y)}$ **torusEllipseW**

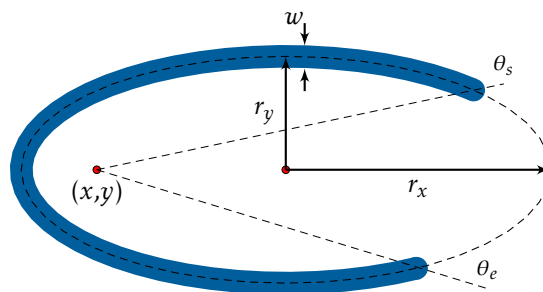


Figure 45: Elliptical torus-arc structure with the center defined at the focus (x, y) .

x y r_x r_y w θ_s θ_e N_{sides} $\theta_{(x,y)}$ **torusEllipseFocusW**

0.4.4 Junctions

0.4.4.1 Cross Junction

x y d w L_1 L_2 r N_{sides} $\theta_{(x,y)}$ `crossJunctionV1`

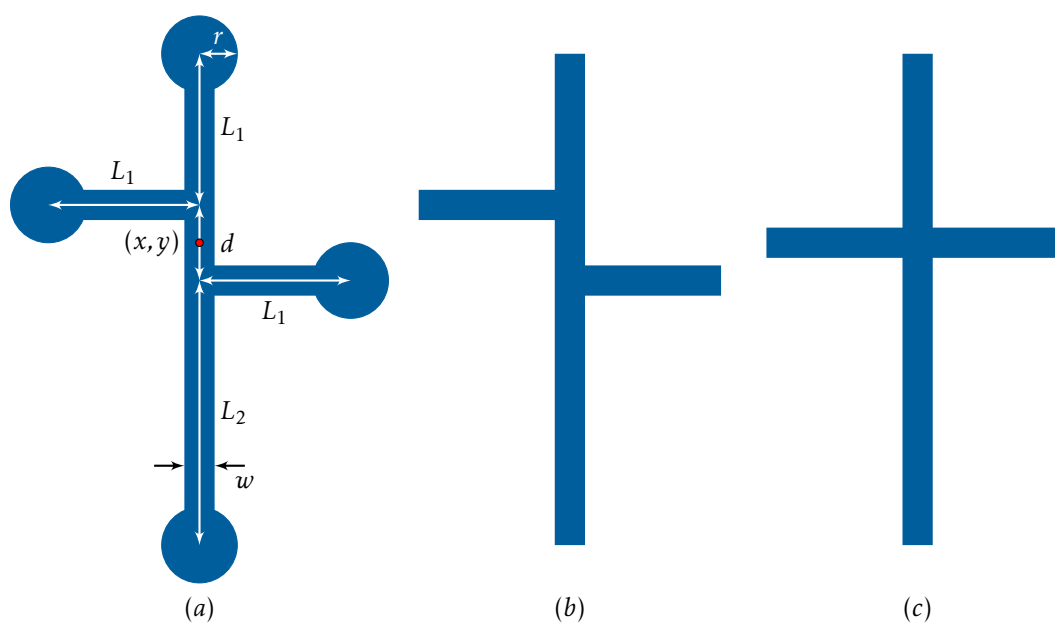


Figure 46: Cross junction with (a) circular reservoirs, (b) $r = 0$ and (c) $r = d = 0$.

x y d w L_1 L_2 L_3 L_4 r N_{sides} $\theta_{(x,y)}$ `crossJunctionV2`

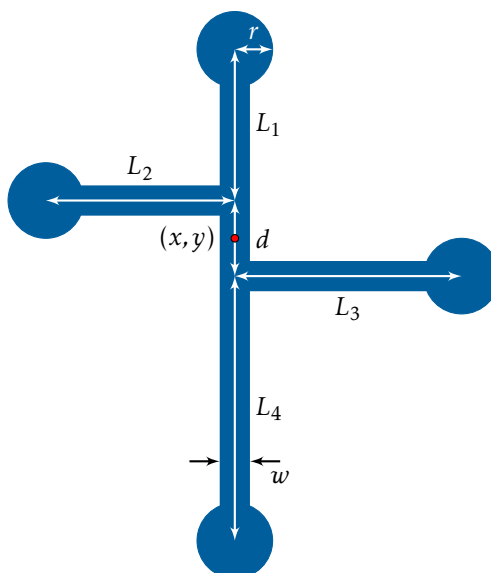


Figure 47: Cross junction with circular reservoirs defined by lengths L_1 , L_2 , L_3 , and L_4 . Setting $r = 0$ eliminates the circular reservoirs and setting $d = 0$ creates a continuous junction between L_2 and L_3 segments (see Figure 46b and 46c).

x y d w L_1 L_2 L_3 L_4 L_{yr1} L_{yr2} r Δr N_{sides} $\theta_{(x,y)}$ **crossJunctionV3**

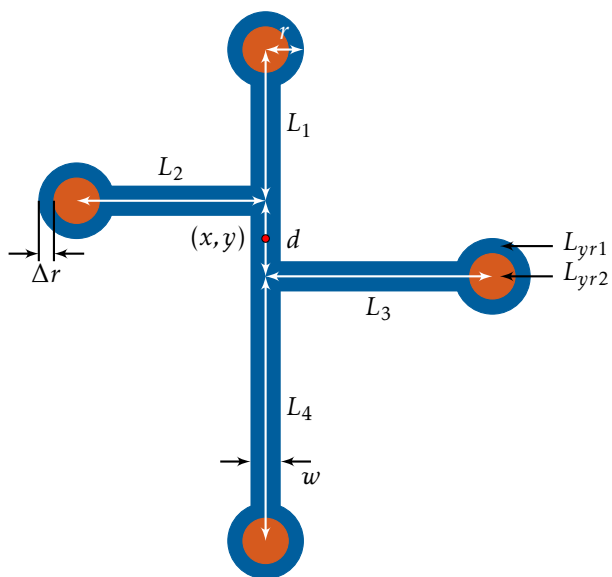


Figure 48: Cross junction similar to **crossJunctionV2** with additional circular reservoirs. L_{yr1} and L_{yr2} are integers (ranging from 0 to 255) representing GDS layers for the structure and the additional circular apertures.

0.4.4.2 Funnel

Fluidic funnel with a rectangular entrance and exit ports.

x y w_{1a} w_{1b} L_{1a} L_{1b} $\theta_{(x,y)}$ **funnel**

x y w_{1a} w_{1b} L_{1a} L_{1b} L_{1r} $\theta_{(x,y)}$ **funnelR**

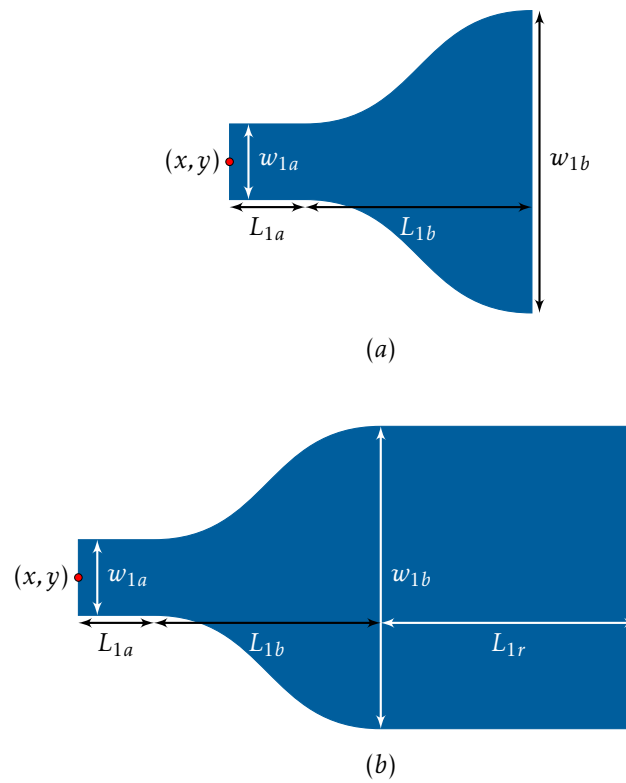


Figure 49: Funnel structures (a) **funnel** and (b) **funnelR**.

0.4.4.3 Funnel Junction

x y w_{1a} w_{1b} L_{1a} L_{1b} w_{2a} w_{2b} L_{2a} L_{2b} w_{3a} w_{3b} L_{3a} L_{3b} w_{4a} w_{4b} L_{4a} L_{4b} $\theta_{(x,y)}$ `funnelJunctionV1`

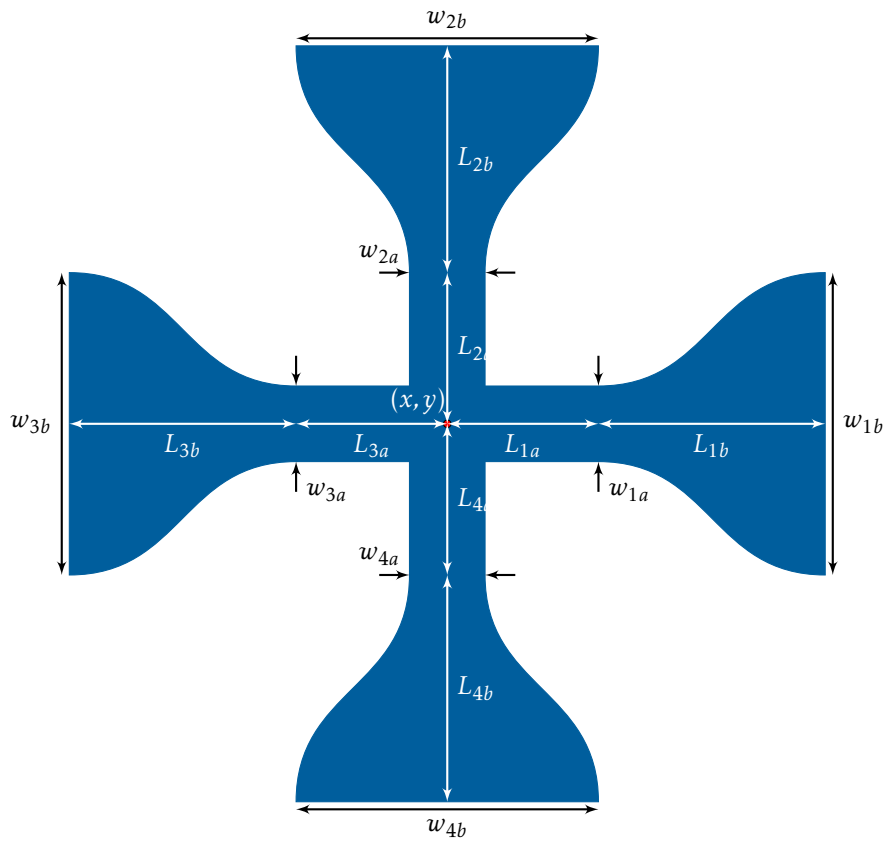


Figure 50: `funnelJunctionV1` structure.

Below constructor has many parameters and is depicted over 2 lines. In general, this constructor could be cast in one continuous CNST script line.

```
x y w1a w1b L1a L1b L1r w2a w2b L2a L2b L2r w3a w3b L3a L3b L3r
w4a w4b L4a L4b L4r  $\theta_{(x,y)}$  funnelJunctionV1R
```

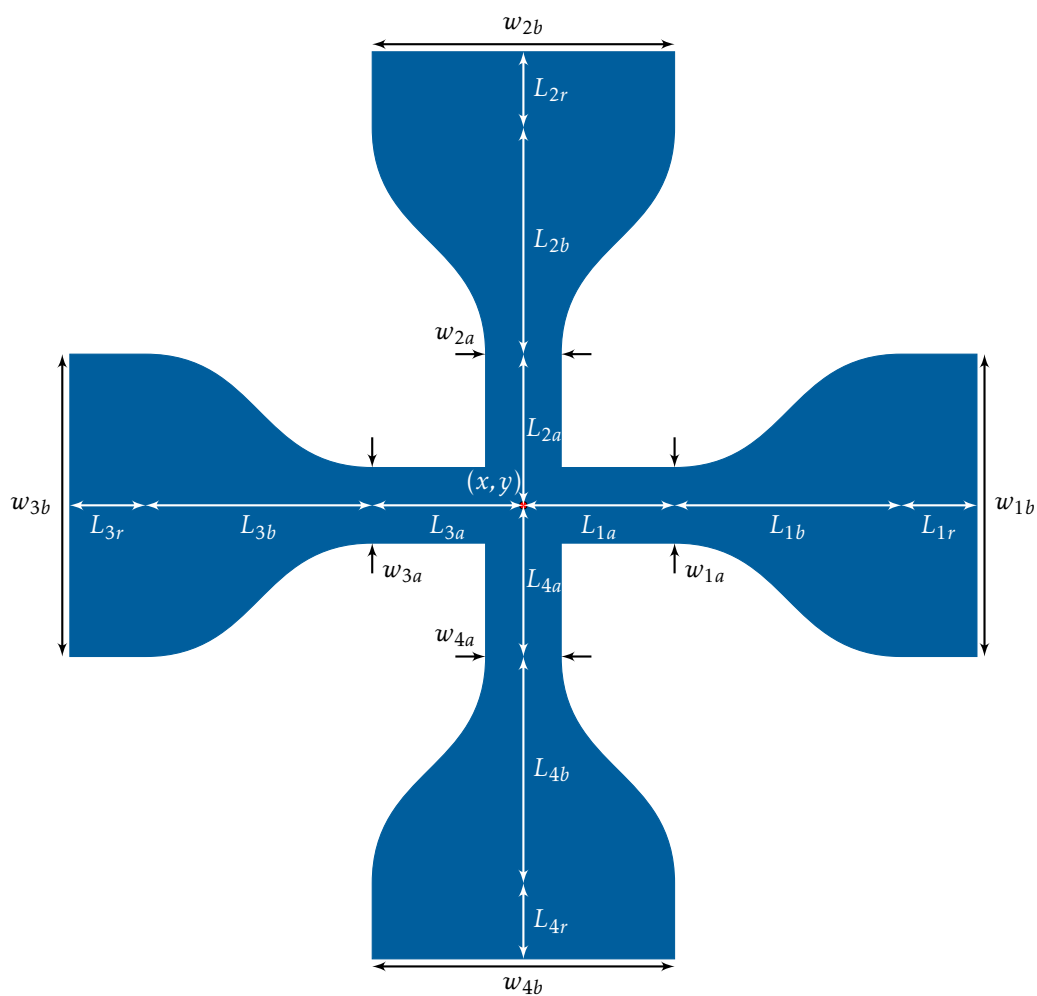


Figure 51: *funnelJunctionV1R* structure.

x y w_1 L_{1a} L_{1b} w_2 L_2 w_3 L_{3a} L_{3b} $\theta_{(x,y)}$ `funnelJunctionV2`

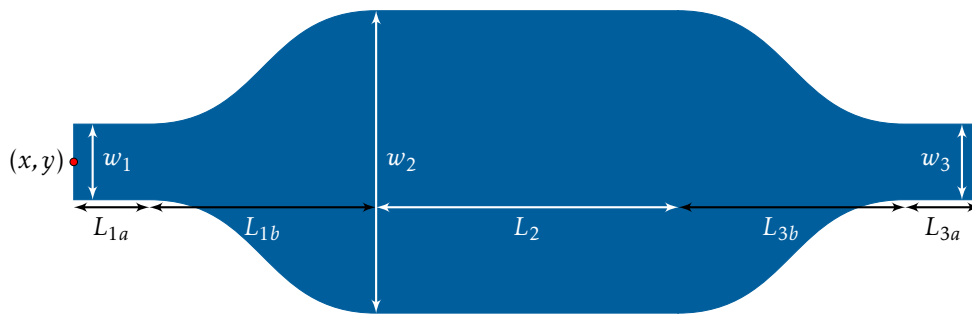


Figure 52: `funnelJunctionV2` structure.

x y w_1 L_{1a} L_{1b} w_2 L_2 w_3 L_{3a} L_{3b} $\theta_{(x,y)}$ `funnelJunctionV2C`

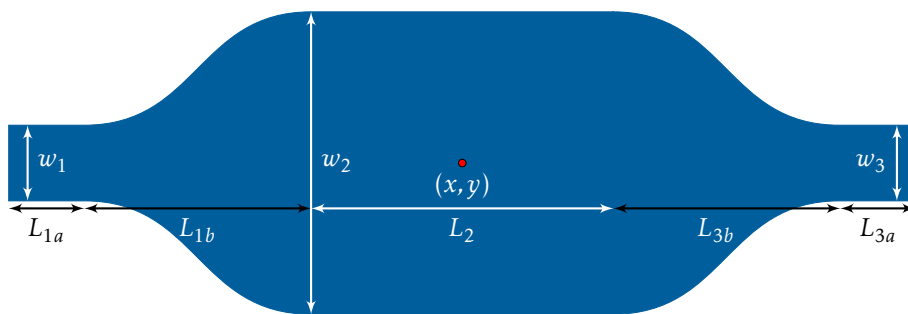


Figure 53: `funnelJunctionV2C` structure with origin (x,y) defined at the center of the junction.

0.4.5 Radial Fill Along An Arc

Radial fill constructor is used to create liquid channel posts around an arc defined by a start and end angles θ_s and θ_e , respectively. The arc is centered at (x, y) , with a radius r , width w , distance from edge d_e . The array of dots of radius r_c are positioned at a distance d from the arc edge. N_{sides} parameter defines the number of each dot arrayed with a pitch Δ . `uniqueStructName` is a string that defines the GDS structure for the circular dot.

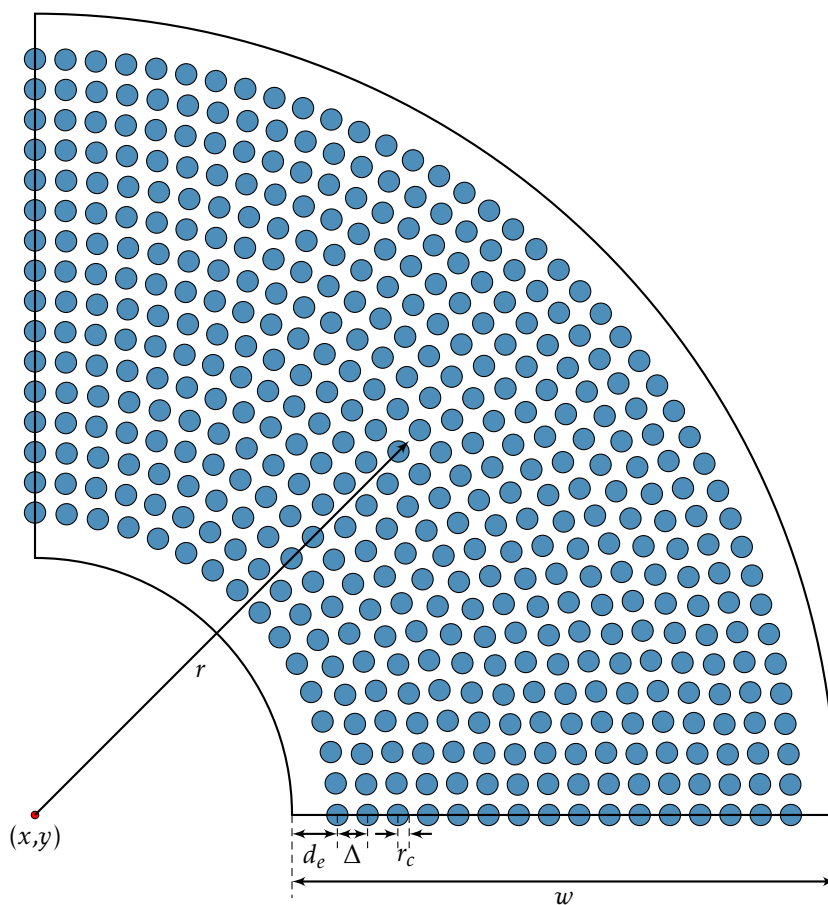


Figure 54: Radial array of dots within an arc segment.

`<uniqueStructName x y r w r_c N_sides Δ d_e θ_s θ_e arcRadialFill>`

0.4.6 Radially distributed circles of varying radii

The two constructors create radially distributed polygon (or circle for vectorized shapes) of radius r_i along a circumference defined by $2\pi R_i$ at a pitch of dr_i , where the integer $i = 1, 2, 3, 4 \dots n$. The radial pattern is distributed around a central circle of radius r_o is placed at (x, y) . If $2\pi R_i/dr_i$ is not an integer, then along a particular R_i the distance from the last circle to the first will not equal dr_i . Next section defined a similar object that distributes the circular objects uniformly around a circumference of radius R_i . N_i defines the number of vertices of the r_i polygon element defined by the **radialCircle** constructor. **shapeReso** defines the rendering resolution of the individual elements of the the vectorized constructor **radialCircleV**.

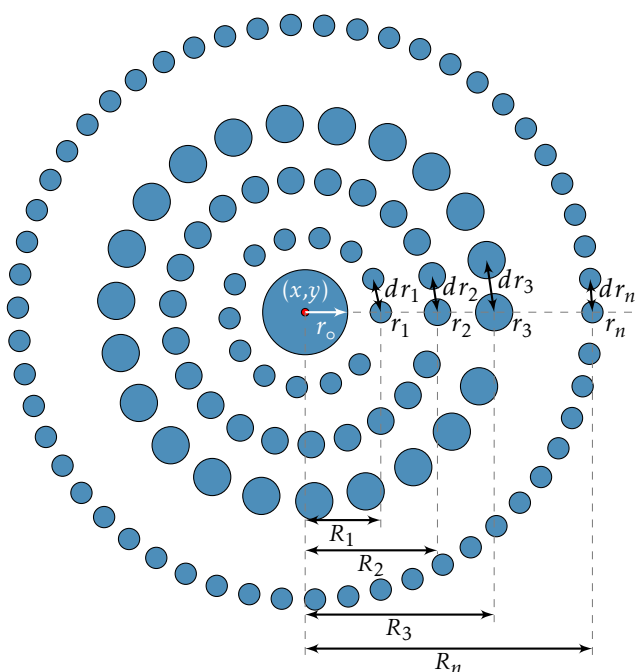


Figure 55: Radial array of circles around a circumference defined by series of radii R_i at a pitch dr_i .

A new constructor **radialCircleName** is introduced to hold the Struct name of the individual circular dots of radius r_i . **uniqueStructName** is a string that defines the prefix GDS structure for the circular dot. Various parameters are appended to this string, i.e. radius, layer, and angle of rotation.

`<uniqueStructName radialCircleName>`

In the following two constructors all the circular elements are cast to a distinct layer defined by the **layer** parameter. As shown in figure 56, within the following constructors parameters can be distributed over multiple lines.

```
x y ro No R1 r1 dr1 N1 ... Rn rn drn Nn radialCircle
x y ro      R1 r1 dr1      ... Rn rn drn      radialCircleV
```

The following constructor allows for attributing layer L_i , data type D_i and rotation θ_i to individual structures defined by r_i .

```
x y ro No Lo Do θo R1 r1 dr1 N1 L1 D1 θ1 ... Rn rn drn Nn Ln Dn θn radialCircle2
x y ro      Lo Do θo R1 r1 dr1      L1 D1 θ1 ... Rn rn drn      Ln Dn θn radialCircleV2
```

```
<radialCircleExample struct>
4 layer
<circRad radialCircleName>
0 0
0.3 40
1.0 0.16 0.46 44
1.75 0.1 0.49 46
2.5 0.2 0.7 50
3.8 0.1 0.44 52 radialCircle
```

Figure 56: Multiline **cnst** script file example using the **radialCircle** constructor.

The following constructors are similar to the previous two with the exception that the circles of radius r_i are uniformly distributed along a circumference $2\pi R_i$ at a pitch of $dr'_i = dr_i + \delta$, where the integer $i = 1, 2, 3, 4 \dots n$. To avoid a non-uniform gap between the last and first circular elements along a distinct circumference at R_i , the user defined pitch dr_i is modified by δ to ensure a uniform distribution of elements along the R_i periphery.

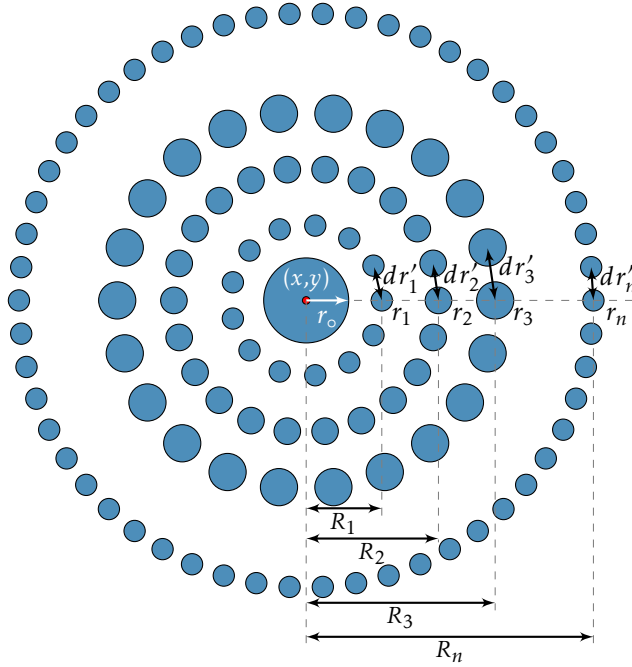


Figure 57: Radial array of circles around a circumference defined by series of radii R_i at a pitch $dr'_i = dr_i + \delta$.

$x \ y \ r_o \ N_o \ R_1 \ r_1 \ dr_1 \ N_1 \ \dots \ R_n \ r_n \ dr_n \ N_n$ **radialCircleU**

$x \ y \ r_o \ \quad R_1 \ r_1 \ dr_1 \ \quad \dots \ R_n \ r_n \ dr_n$ **radialCircleUV**

The following constructor allows for attributing layer L_i , data type D_i and rotation θ_i to individual structures defined by r_i .

$x \ y \ r_o \ N_o \ L_o \ D_o \ \theta_o \ R_1 \ r_1 \ dr_1 \ N_1 \ L_1 \ D_1 \ \theta_1 \ \dots \ R_n \ r_n \ dr_n \ N_n \ L_n \ D_n \ \theta_n$ **radialCircleU2**

$x \ y \ r_o \ \quad L_o \ D_o \ \theta_o \ R_1 \ r_1 \ dr_1 \ \quad L_1 \ D_1 \ \theta_1 \ \dots \ R_n \ r_n \ dr_n \ \quad L_n \ D_n \ \theta_n$ **radialCircleUV2**

0.4.7 Radial Arrays

In contrast to the **radialCircle** constructors (section 0.4.6), radial array methods presented in this section are created using shapes with a specified number of vertices (N_{sides}). There are two constructors for each radial array shape, one which creates a quarter of an array and the second (constructor with a **W** suffix) which creates the entire structure. For the former, to preserve symmetry of the individual rings, the $\frac{1}{4}$ of the array is created from instantiated individual shapes and then stored into a GDS struct. The quarter segment is then instantiated to create the full structure within the active GDS struct. This constructor is not suited for non-symmetric shapes. The `uniqueStructPrefix` string is used to create two additional GDS structs that hold the individual objects and the instantiated $\frac{1}{4}$ of the array. For the structs, the radii, rotation and number of vertices are appended to the struct name, with an additional string `_QS` for the quarter shape struct. Constructors ending with a **W** are constructed from an individual shape that is instantiated across each ring spanning 2π and should be used with non-symmetric shapes. Since the string defining a GDS structure has a character limit, the `uniqueStructPrefix` parameter should be kept short.

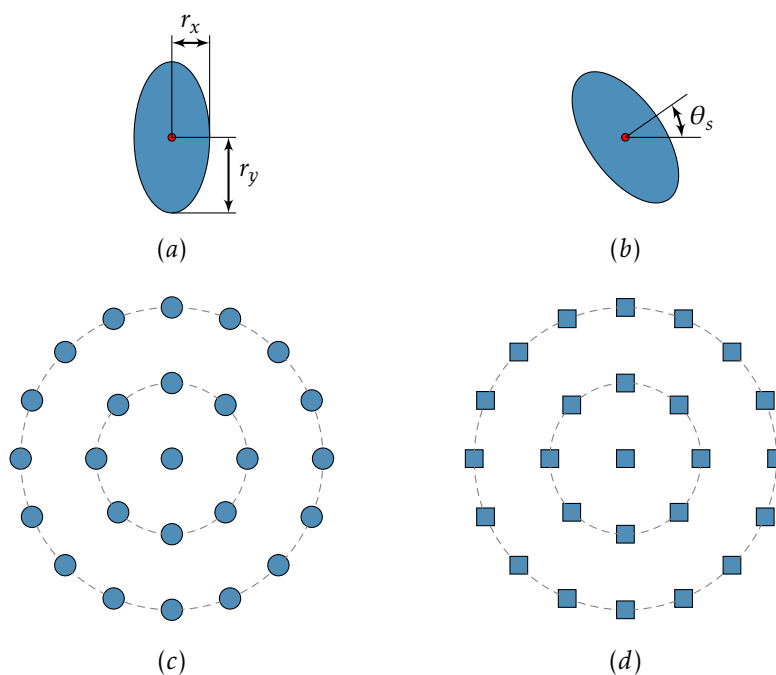


Figure 58: Shapes with dimensions of r_x and r_y at (a) $\theta = 0$ and (b) $\theta = 35^\circ$. Radial array of shapes with $r_x = r_y$, (c) $N_{sides} = 44$ and (d) $N_{sides} = 4$.

0.4.7.1 Variable ring position and shape pitch

The **radialArray** constructor creates a user defined radial array of shapes centered around (x, y) and distributed along radii R_1, R_2, \dots, R_n at a respective pitch of p_1, p_2, \dots, p_n . The individual shapes are defined by radii r_{x_i} and r_{y_i} with a number of vertices N_{v_i} , where integer $i = 0, 1, 2, \dots$ is prescribed to rows of shapes. The first array element ($i = 0$) is placed at position (x, y) . The next row of arrayed elements ($i = 1$) is positioned at a relative distance R_1 from (x, y) . The process continues until the array element $i = n$.

$x \ y \ r_{x_0} \ r_{y_0} \ \theta_0 \ N_{v_0} \ R_1 \ r_{x_1} \ r_{y_1} \ \theta_1 \ N_{v_1} \ p_1 \ \dots \ R_n \ r_{x_n} \ r_{y_n} \ \theta_n \ N_{v_n} \ p_n$ **radialArray**

$x \ y \ r_{x_0} \ r_{y_0} \ \theta_0 \ N_{v_0} \ R_1 \ r_{x_1} \ r_{y_1} \ \theta_1 \ N_{v_1} \ p_1 \ \dots \ R_n \ r_{x_n} \ r_{y_n} \ \theta_n \ N_{v_n} \ p_n$ **radialArrayW**

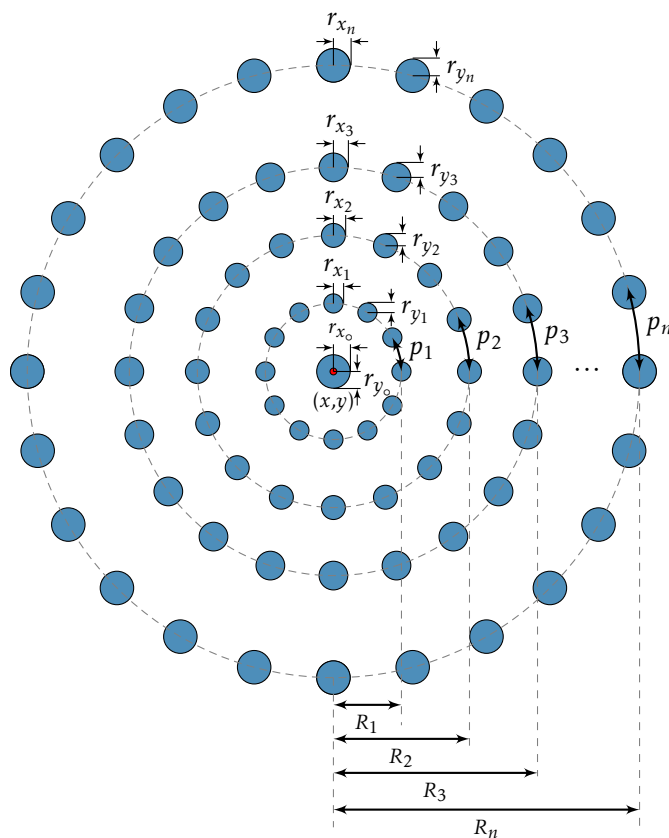


Figure 59: Radial array of shapes.

A new constructor `structPrefix` is introduced to hold the Struct name of individual shapes. `uniqueStructPrefix` is a string that defines the prefix of the GDS structure of a particular shape i . The ring radius (R_i), shape radii (r_{x_i} and r_{y_i}), rotation (θ_i), number of vertices (N_{v_i}), and pitch (p_i). Since the string defining a GDS structure has a 32 character limit, the `uniqueStructPrefix` parameter should be kept short. The default value of `uniqueStructPrefix` is an empty string, hence if it's not defined, the GDS struct naming convention for individual elements and quarter shapes will be with an underscore.

`<uniqueStructPrefix structPrefix>`

```
<00AradialArray struct>
<a1RA structPrefix>
2 -4 0.1 0.1 0 22
1.5 0.15 0.15 0 22 0.4
3 0.2 0.2 0 22 0.5
4.5 0.25 0.25 0 22 0.7
6 0.4 0.4 0 22 1.2
radialArray
#
<00BradialArrayW struct>
<a2RAW structPrefix>
2 -4 0.1 0.1 0 22
1.5 0.15 0.15 0 22 0.4
3 0.2 0.2 0 22 0.5
4.5 0.25 0.25 0 22 0.7
6 0.4 0.4 0 22 1.2
radialArrayW
```

Figure 60: Multiline `cnst` script file example using the `structPrefix`, `radialArray` and `radialArrayW` constructors.

0.4.7.2 Constant ring pitch

Radial array of a shape defined by radii r_x and r_y , with N_{sides} vertices at a pitch of $\approx p_s$ along rings distributed at a constant pitch of p_R . The object pitch p_s is modified by a value that distributes the shapes uniformly about the circumference of the rings. R_{out} defines the boundary of the outermost ring.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_R R_{out} radialArrayConst>

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_R R_{out} radialArrayConstW>

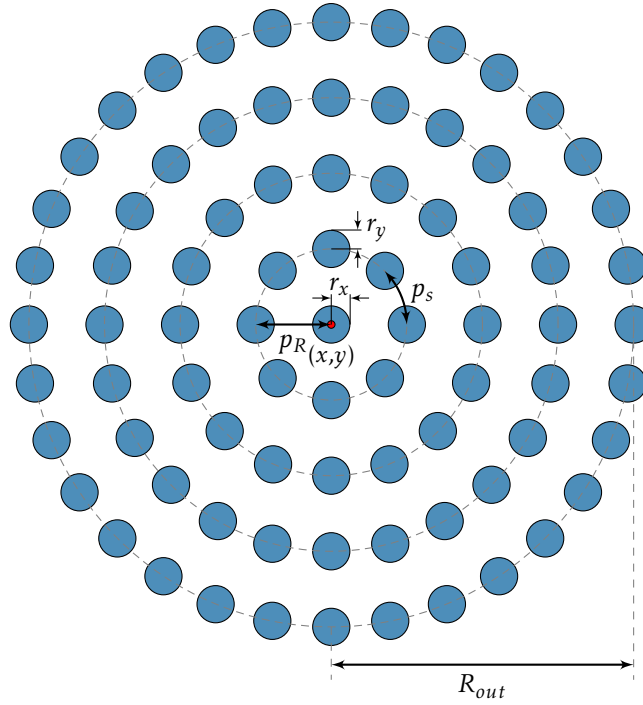


Figure 61: Radial array of shapes distributed over a constant ring pitch p_R .

0.4.7.3 Linearly varying ring separation

Radial array shown below in figure 62 has a linearly varying ring pitch p_R . The pitch increases by the amount Δ with each outward ring.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_R Δ R_{out} radialArrayLin>

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_R Δ R_{out} radialArrayLinW>

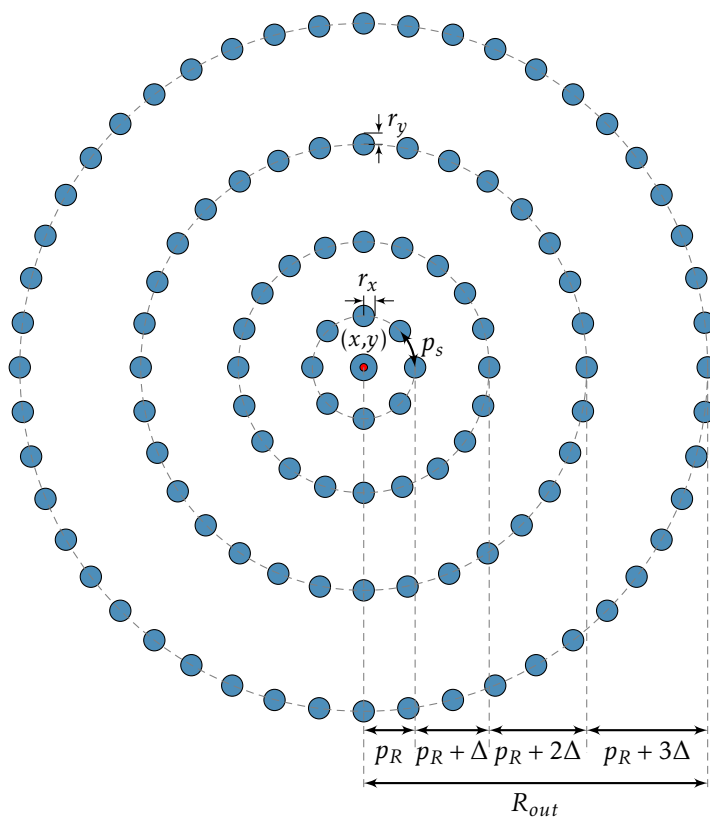


Figure 62: Radial array of shapes with linearly varying ring separation.

Radial array shown below in figure 63 has a linearly varying ring pitch p_R . The pitch increases by the amount Δ with each outward ring, reaching a maximum at $N_{rings}/2$, and then decreases by Δ .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_R Δ N_{rings} radialArrayLin2>

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_R Δ N_{rings} radialArrayLin2W>

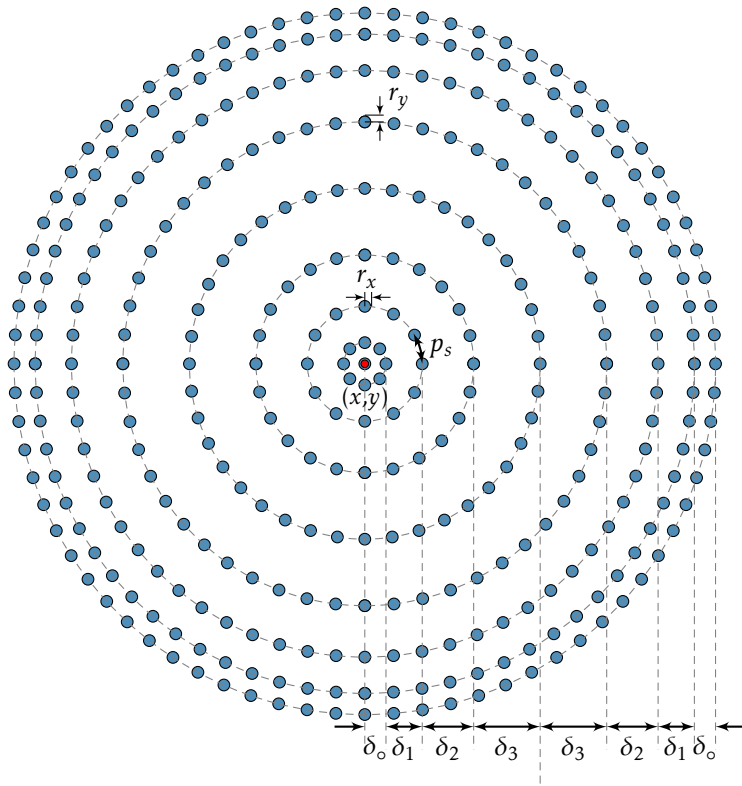


Figure 63: Radially symmetric array of shapes with linearly varying ring separation, where $\delta_i = p_R + i\Delta$

0.4.7.4 Exponentially varying ring separation

The below constructor creates a radial array of shapes with exponentially varying ring separation. The pitch increases exponentially between the values of p_{start} and p_{end} for N_{rings} number of rings, as shown in figure 64.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_{start} p_{end} N_{rings} radialArrayExp>

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_{start} p_{end} N_{rings} radialArrayExpW>

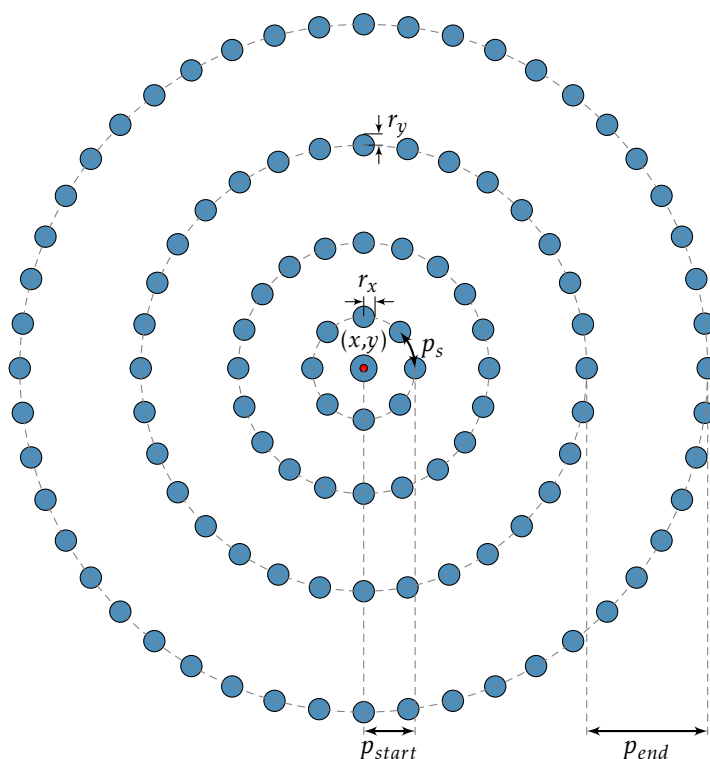


Figure 64: Radial array of shapes with exponentially varying ring separation between pitch start and end values, p_{start} and p_{end} , respectively.

The below constructor creates a radial array of shapes with exponentially varying ring separation. The pitch, first increases, then decreases exponentially between the values of p_{start} and p_{end} for N_{rings} number of rings, as shown in figure 65.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_{start} p_{end} N_{rings} radialArrayExp2>

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_s p_{start} p_{end} N_{rings} radialArrayExp2W>

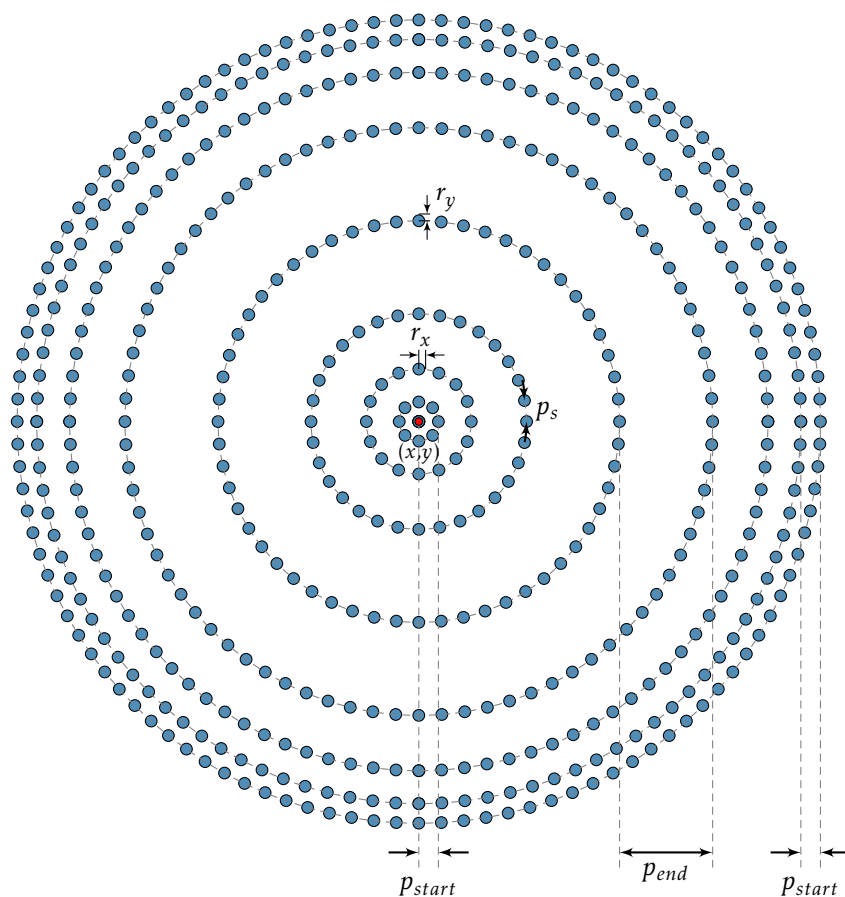


Figure 65: Radial array of shapes with exponentially varying ring separation between pitch start and end values, p_{start} and p_{end} , respectively.

0.4.8 Random Radial Arrays with SVG Export

This section describes constructors that create a random distribution of ellipses within a circular area of radius R . The elliptical shapes, defined by a radii between a minimum and maximum radii (r_{min} and r_{max}), are randomly rotated. The potential P defines the minimum separation between the randomly distributed ellipses in units of micrometers. Each generated shape then is cast into a randomly chosen GDS layer between values L_{min} and L_{max} .

The procedure initiates by creating a circle of radius R with number of vertices and GDS layer defined by N_R and L_R , respectively. A random coordinate (x, y) within R , random ellipse radii r_x and r_y ($r_{min} \leq r_x, r_y \leq r_{max}$) and random rotation about (x, y) ($0 \leq \theta_{x,y} \leq 180$) are then calculated. A vectorized elliptical shape is then created with vertices defined by the `shapeReso` parameter. The shape then undergoes a check to determine if it is completely contained within the circle of radius R . If the conditions are met, the shape is placed within the circular boundary of radius R and new elliptical shape with dimensions $r'_x = r_x + P$ and $r'_y = r_y + P$, number of vertices N_r and equivalent rotation is subtracted from the large circular boundary. The process continues until number of elements (N_E) is reached. For subsequent random shapes, an additional check is performed to determine if the potential P between elements is violated. If either of two conditions, shape not fully contained within R or potential, are violated, a new set of coordinates and dimensions are calculated and tested. This random generation process continues until N_E or the number of iterations (I) is reached.

The resulting GDS file contains the randomly distributed shapes and the remnants of the circular boundary in layer L_R (Figure 66). Computational time can be reduced by choosing smaller values for N_R , N_r , N_E and I .

```
<x y r_min r_max P N_E I R N_r N_R L_min L_max L_R randomRadial>
```

In addition to creating the random shapes within a defined circular shape, the below constructor additionally exports solid and outlined random shapes within the SVG file format (Figure 67). The SVG file name (SVGfileName), without the '.svg' extension, must be constructed from a continuous set of characters. White characters (spaces and tabs) are not allowed. The '.svg' extension will be automatically appended to the SVGfileName string. The constructor has an additional parameter w_t that defines the thickness of the outlined SVG shapes. For the outlined shapes, the phrase Outline.svg will be appended to the SVGfileName string.

```
<x y r_min r_max P N_E I R N_r N_R L_min L_max L_R w_t SVGfileName randomRadial>
```

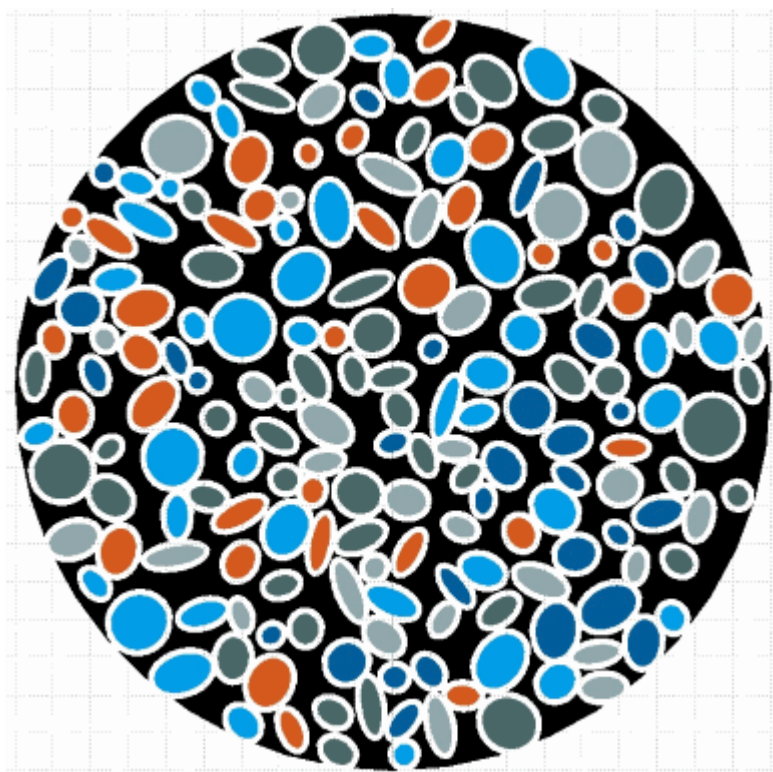
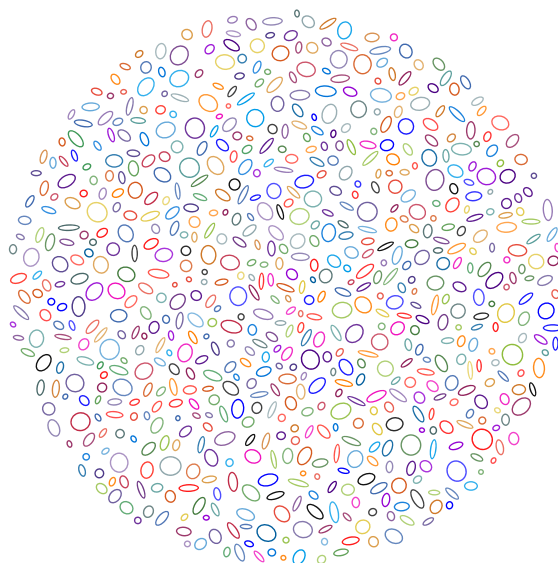


Figure 66: *randomRadial* example of a rendered GDS output. The remaining circular boundary of radius R is showing in black.



(a)



(b)

Figure 67: *randomRadialSVG* example of a (a) solid and (b) outline SVG output.

0.4.9 Non-uniform Arrays

Arrays of shapes with specified number of vertices (N_{sides}) are created using the presented methods within this section. The `uniqueStructPrefix` string is used to create a GDS stuct that holds the individual shape objects of the array. The shape radii, rotation and number of vertices parameters are appended to this string. Since the string defining a GDS structure has a 32 character limit, the `uniqueStructPrefix` parameter should be kept short.

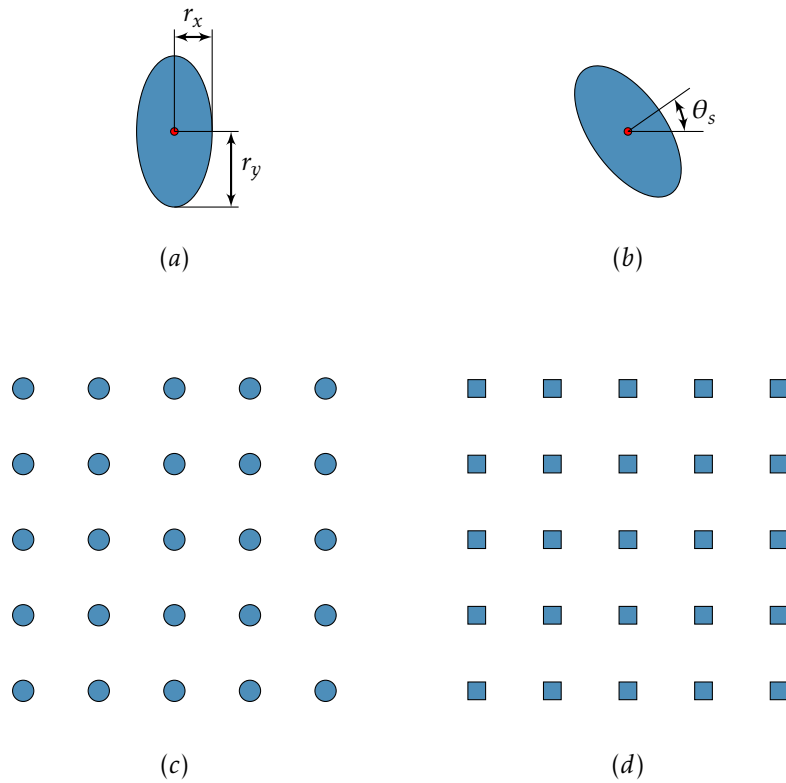


Figure 68: Shapes with dimensions of r_x and r_y at (a) $\theta = 0$ and (b) $\theta = 35^\circ$. Array of shapes with $r_x = r_y$, (c) $N_{sides} = 44$ and (d) $N_{sides} = 4$.

0.4.9.1 Generalized non-uniform array

The generalized non-uniform array constructor (**NUA**) creates a user defined rectangular array. The shapes are defined by radii r_{x_i} and r_{y_i} with a number of vertices N_{v_i} , where integer $i = 0, 1, 2, \dots$ is prescribed to rows of shapes. The first array element ($i = 0$) is placed at position (x, y) . N_{s_o} is number of arrayed polygon shapes placed at a pitch dp_o . The next row of arrayed elements ($i = 1$) is positioned at a relative distance p_1 from (x, y) . The process continues until the array element $i = n$.

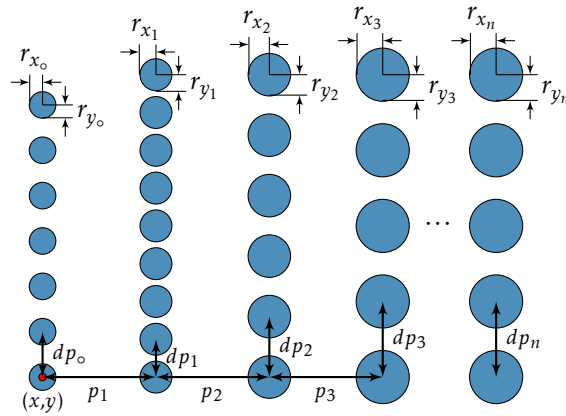
$$\begin{array}{cccccccccccccccc}
 x & y & r_{x_o} & r_{y_o} & \theta_o & N_{v_o} & dp_o & N_{s_o} & p_1 & r_{x_1} & r_{y_1} & \theta_1 & N_{v_1} & dp_1 & N_{s_1} \\
 & & & & & & & & & & & & & & & \\
 & & & & & & & & \dots & p_n & r_{x_n} & r_{y_n} & \theta_n & N_{v_n} & dp_n & N_{s_n} & \text{NUA}
 \end{array}$$


Figure 69: Generalized non-uniform array **NUA** constructor example.

Similar to the **radialArray** constructor (section 0.4.7.1), the **structPrefix** constructor is used to hold the **Struct** name of individual shapes. Here, **uniqueStructPrefix** is a string that defines the prefix of the GDS structure of a particular shape i . The pitch (p_i), shape radii, rotation, number of vertices, pitch and number of array element parameters are appended to this string. Since the string defining a GDS structure has a 32 character limit, the **uniqueStructPrefix** parameter should be kept short.

<uniqueStructPrefix structPrefix>

0.4.9.2 Linearly varying pitch array

The non-uniform array shown below in figure 70 has a linearly varying pitch p between each column. The pitch increases by the amount Δ with each columnar array. The array extents are defined by S_A . The ends of the array are staggered since pitch is preserved. The following section will introduce a similar object that allows for columns to be aligned to the columnar extent S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALin>

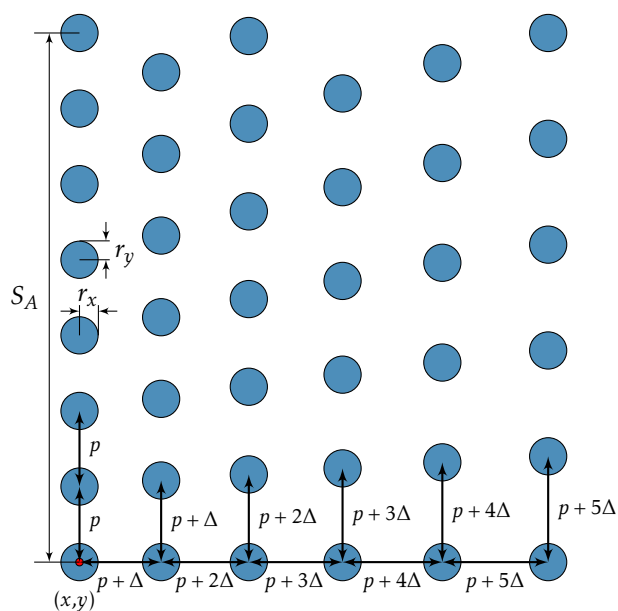


Figure 70: Array of shapes with a linearly varying pitch.

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALinR>

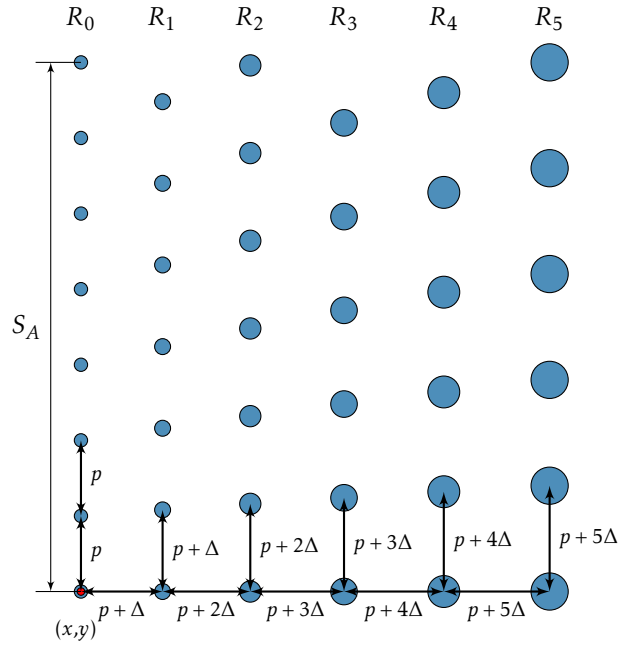


Figure 71: Array of shapes with a linearly varying pitch and radius.

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.3 Linearly varying pitch symmetric array

Similar to the linearly varying pitch array shown in the previous section 0.4.9.2, the below ramped array is composed of two linearly varying pitch arrays. The pattern is a square array of size S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALin2>

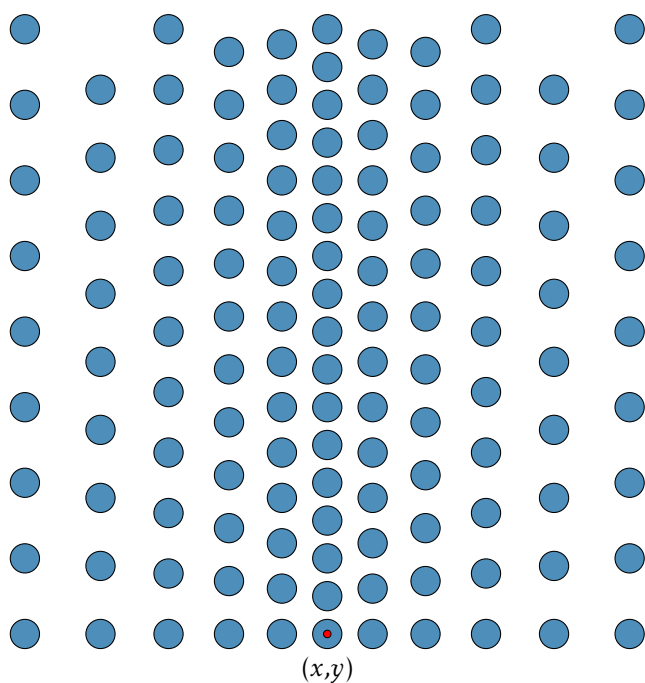


Figure 72: Double ramped array of shapes with a linearly varying pitch.

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALin2R>

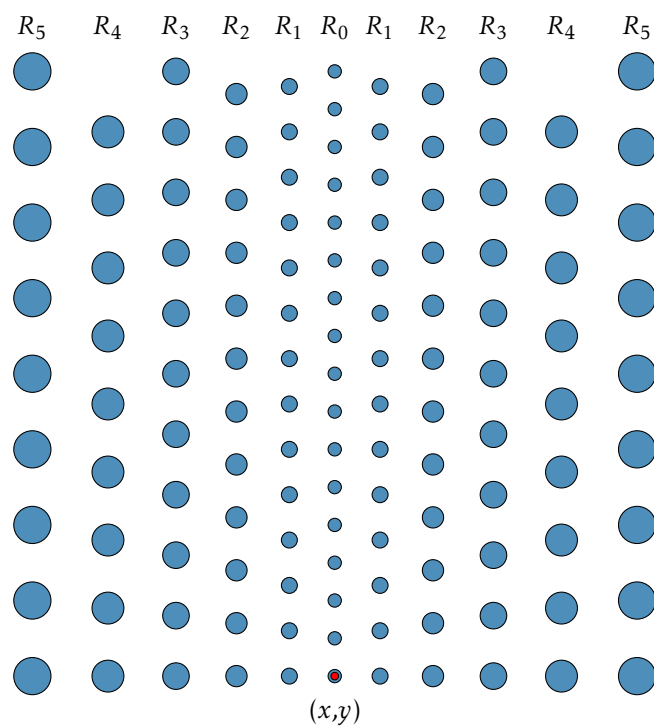


Figure 73: Double ramped array of shapes with a linearly varying pitch and radius.

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

The array is similar to the one in figure 72, with the exception that the horizontal extent is twice S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALin2D>

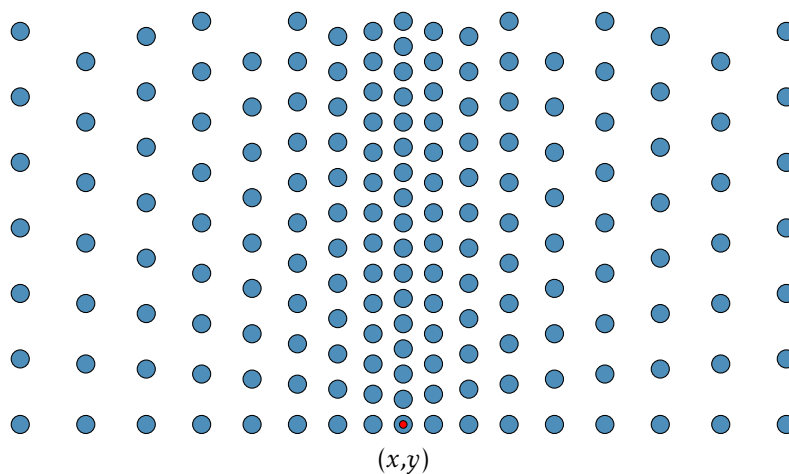


Figure 74: Double ramped array of shapes with a linearly varying pitch where the horizontal extent is $2S_A$.

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALin2DR>

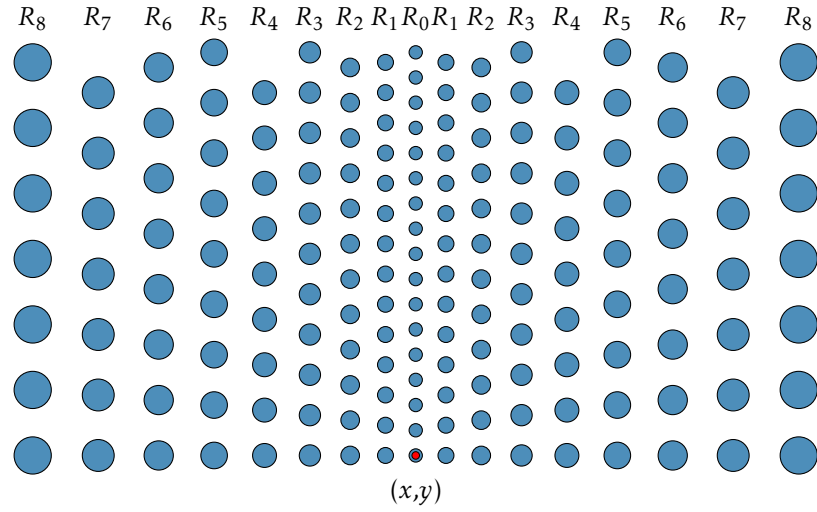


Figure 75: Double ramped array of shapes with a linearly varying pitch and radius where the horizontal extent is $2S_A$.

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.4 Linearly varying pitch aligned array

Similar to the non-uniform array shown in section 0.4.9.2, the constructor **NUALinA** creates an array where pitch between each row element is adjusted so that the end element coincides with the columnar extent S_A . Quantities $\Delta_1, \Delta_2, \Delta_3, \Delta_4 \dots$ represent values that are used to adjust the pitch so that columns align with the array extent.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A **NUALinA**>

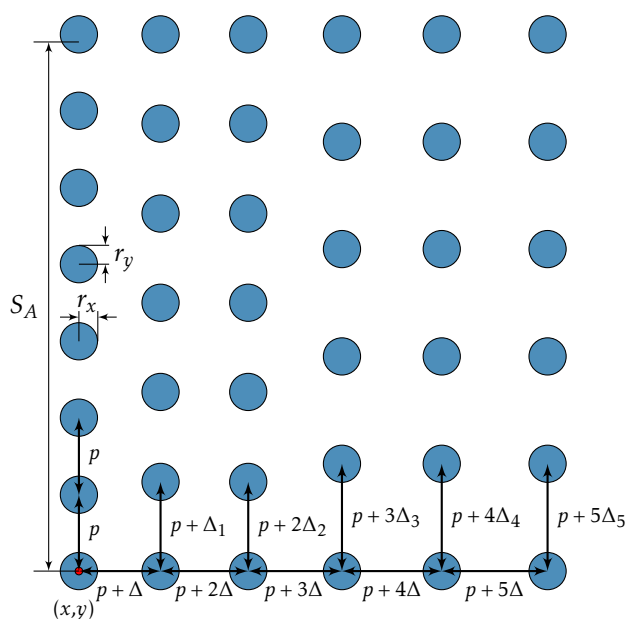


Figure 76: Array of shapes with a linearly varying pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALinAR>

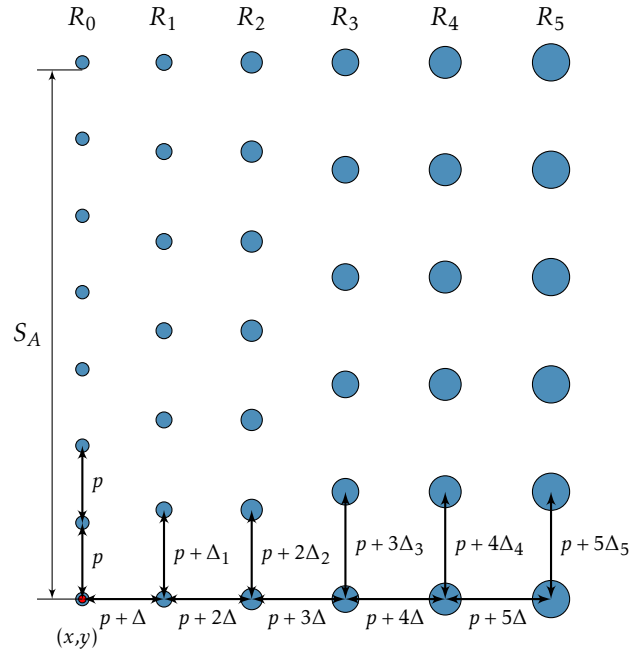


Figure 77: Array of shapes with a linearly varying pitch and radius. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.5 Linearly varying pitch aligned symmetric array

Similar to the aligned linearly varying pitch array shown in the previous section 0.4.9.4, the below ramped array is composed of two aligned, linearly varying pitch arrays. The pattern is a square array of size S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALinA2>

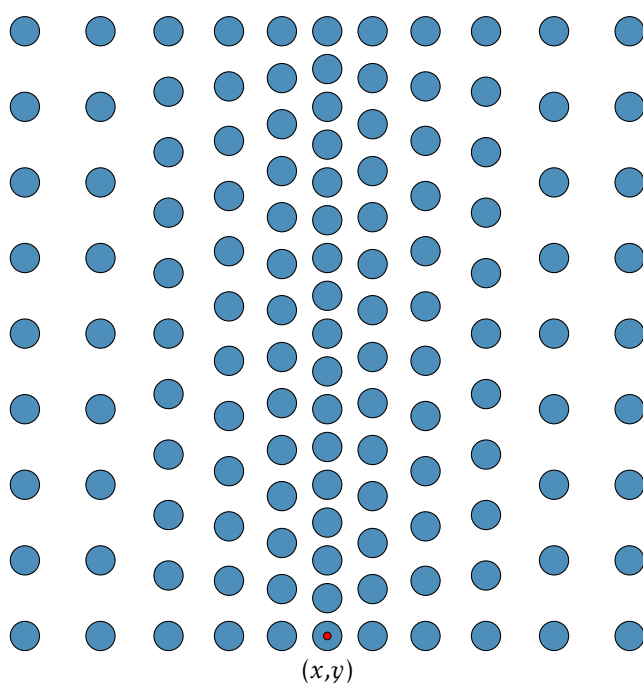


Figure 78: Ramped array of shapes with a linearly varying pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

NUALinA2R>



the array size S_A .

columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

The array is similar to the one in figure 78, with the exception that the horizontal extent is twice S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALinA2D>

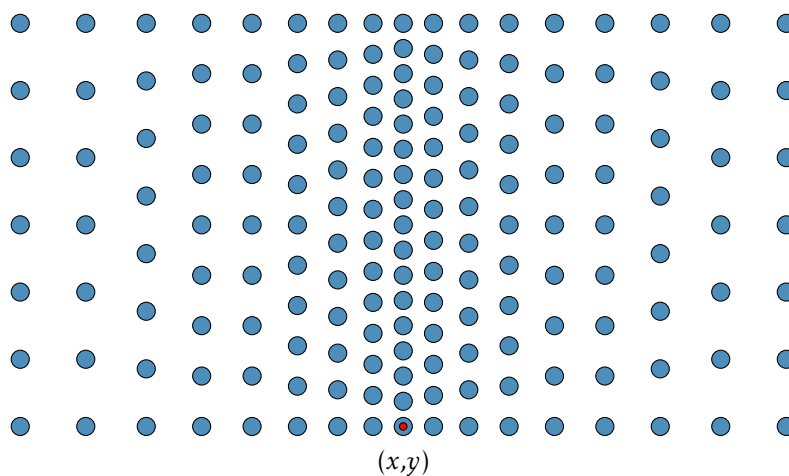


Figure 80: Ramped array of shapes with a linearly varying pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the horizontal extent is $2S_A$.

NUALinA2DR>



the horizontal extent is $2S_A$.

columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.6 Linearly varying vertical and constant horizontal pitch

The below ramped array is composed of arrayed shapes with a linearly varying vertical ($p + i\Delta$ where the positive integer $i = 0, 1, 2, \dots$) and constant horizontal pitch (p). The pattern is a square array of size S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALin3>

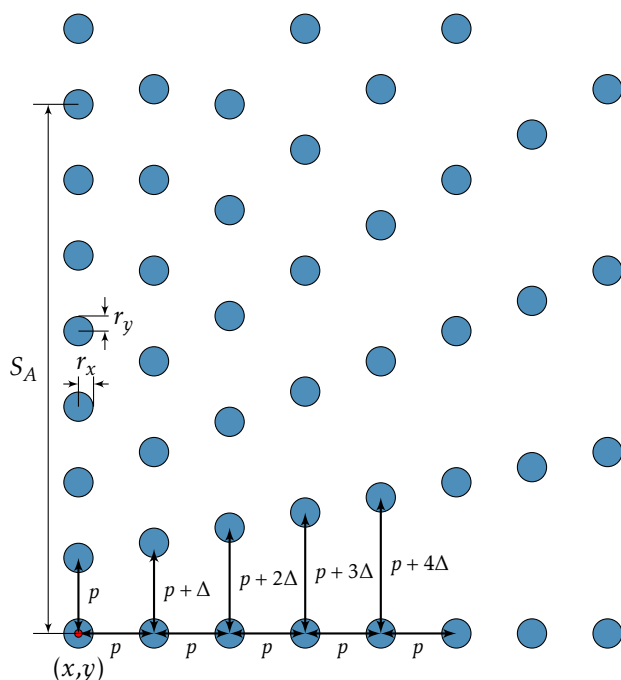


Figure 82: Array of shapes with a linearly varying vertical and constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALin3R>

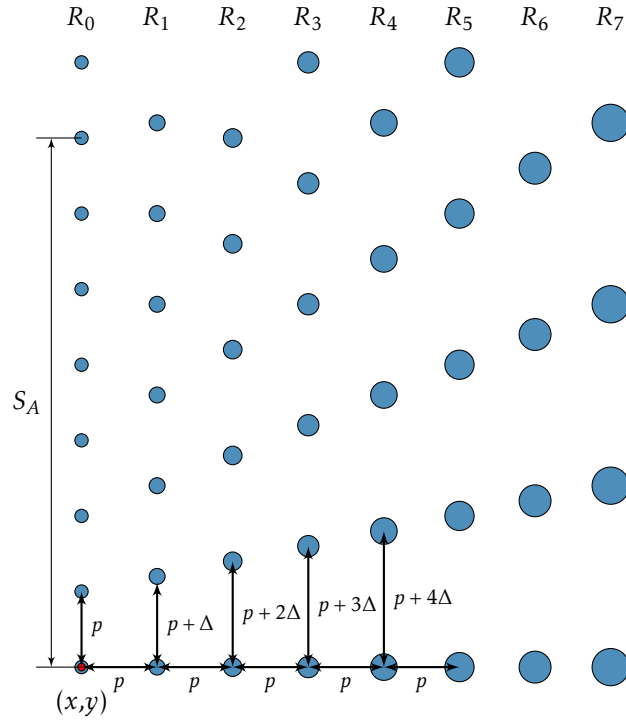


Figure 83: Array of shapes with a linearly varying vertical pitch and radius, and a constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.7 Linearly varying vertical and constant horizontal pitch, aligned array

Similar to the array in section 0.4.9.6, the below aligned array is composed of shapes placed on a linearly varying vertical ($p + i\Delta$ where the positive integer $i = 0, 1, 2, \dots$) and constant horizontal pitch (p). The pattern is a square array of size S_A . Similar to the non-uniform array shown in section 0.4.9.6, the constructor **NUALinA3** creates an array where pitch between each row element is adjusted so that the end element coincides with the columnar extent S_A . Quantities $\Delta_1, \Delta_2, \Delta_3, \Delta_4 \dots$ represent values that are used to adjust the pitch so that columns align with the array extent.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A **NUALinA3**>

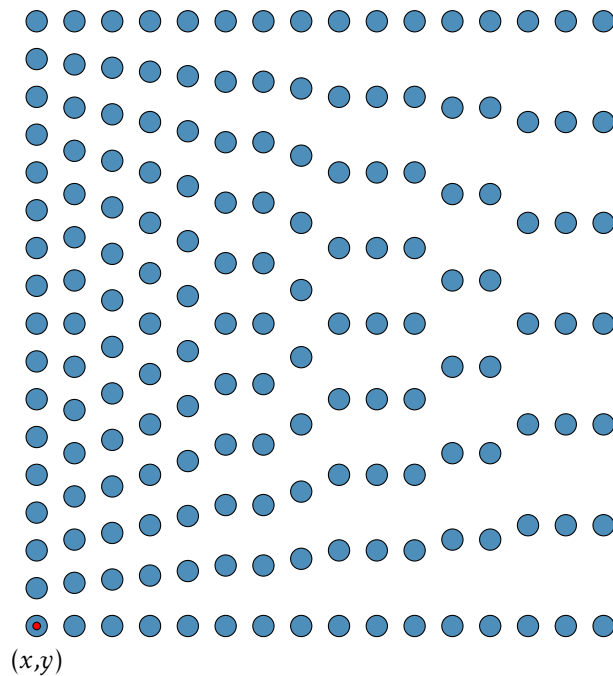


Figure 84: Array of shapes with a linearly varying vertical and constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALinA3R>

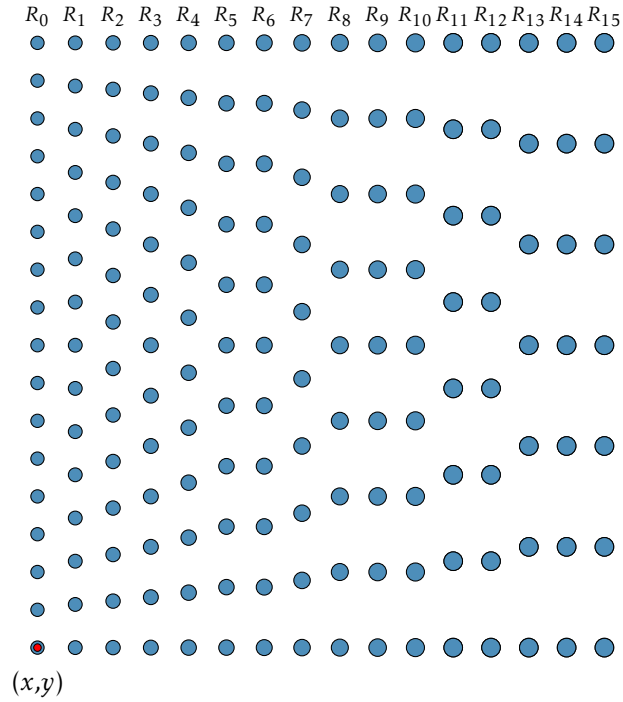


Figure 85: Array of shapes with a linearly varying vertical pitch and radius, and a constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.8 Linearly varying vertical and constant horizontal pitch symmetric array

Similar to the linearly varying pitch array shown in the previous section 0.4.9.6, the below ramped array is composed of two linearly varying vertical ($p + i\Delta$ where the positive integer $i = 0, 1, 2, \dots$) and constant horizontal pitch (p) arrays. The pattern is an array of size S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALin4>

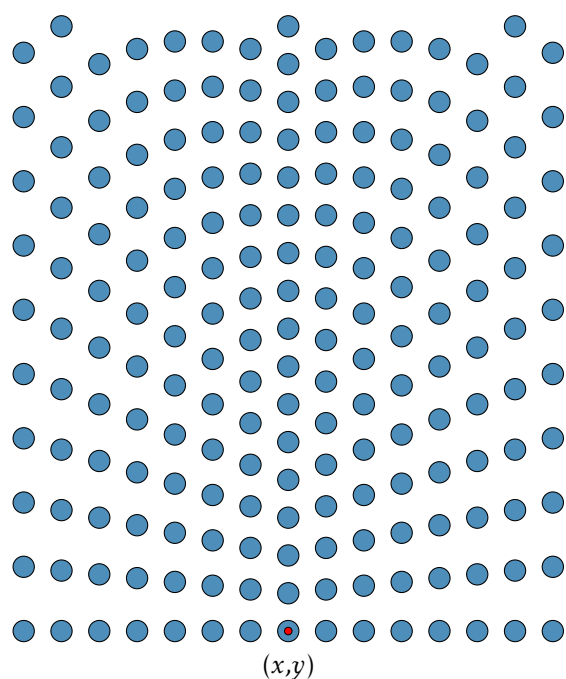


Figure 86: Double ramped array of shapes with a linearly varying vertical and constant horizontal pitch. The horizontal extent is S_A .

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALin4R>

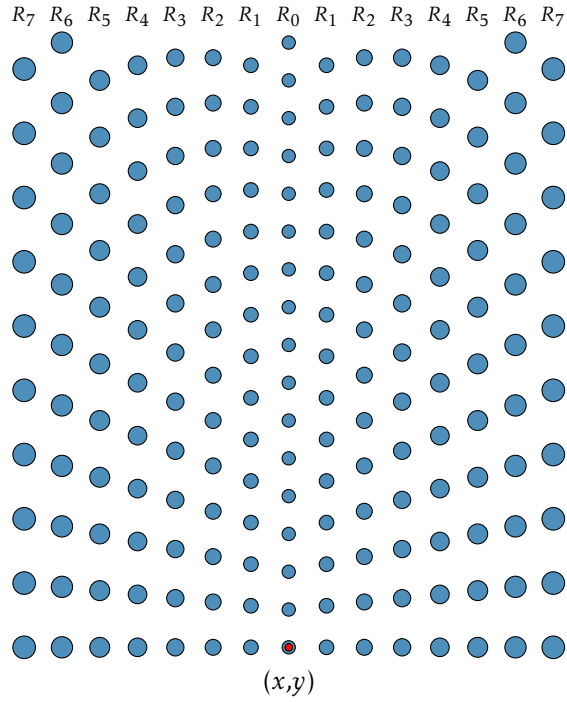


Figure 87: Double ramped array of shapes with a linearly varying vertical pitch and radius, and a constant horizontal pitch. The horizontal extent is S_A .

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

The array is similar to the one in figure 88, with the exception that the horizontal extent is twice S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALin4D>

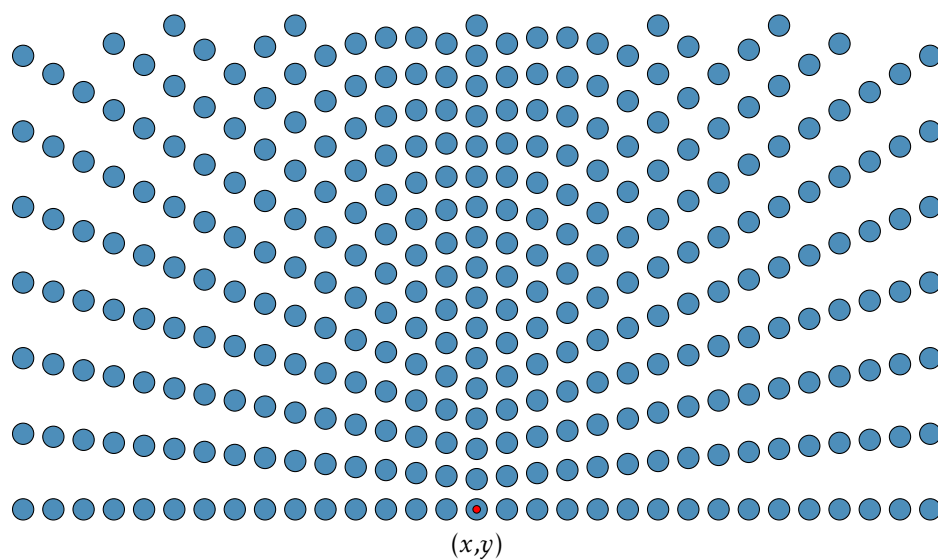


Figure 88: Double ramped array of shapes with a linearly varying vertical and constant horizontal pitch. The horizontal extent is $2S_A$.

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALin4DR>

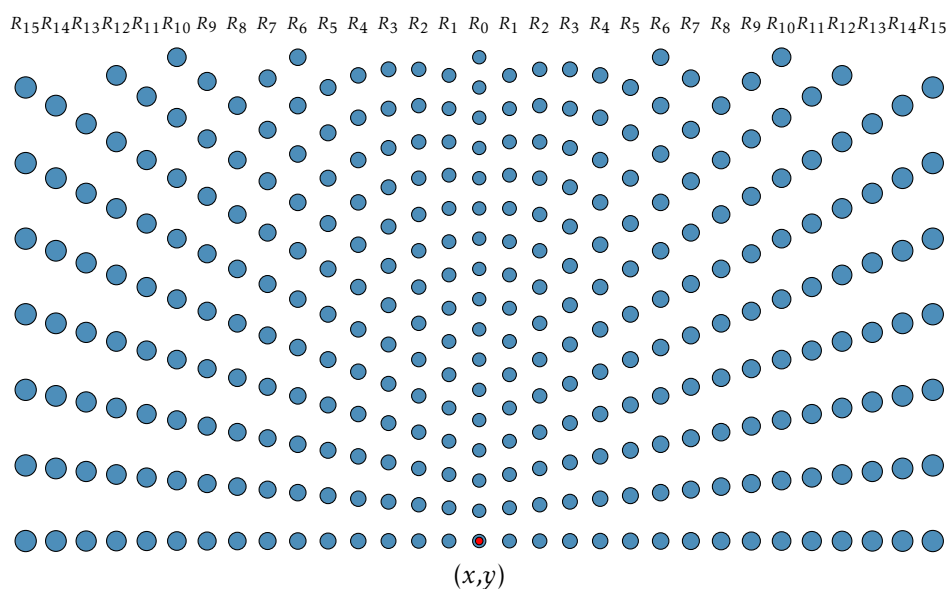


Figure 89: Double ramped array of shapes with a linearly varying vertical pitch and radius, and a constant horizontal pitch. The horizontal extent is $2S_A$.

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.9 Linearly varying vertical and constant horizontal pitch, aligned symmetric array

Similar to the linearly varying pitch array shown in the previous section 0.4.9.6, the below ramped array is composed of two linearly varying vertical ($p + i\Delta$ where the positive integer $i = 0, 1, 2, \dots$) and constant horizontal pitch (p) arrays. The pattern is an array of size S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALinA4>

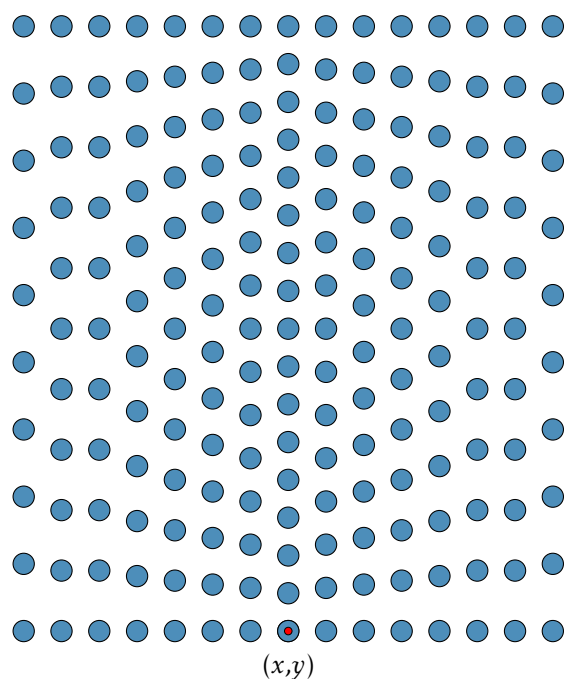


Figure 90: Double ramped array of shapes with a linearly varying vertical and constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the horizontal extent is S_A .

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALinA4R>

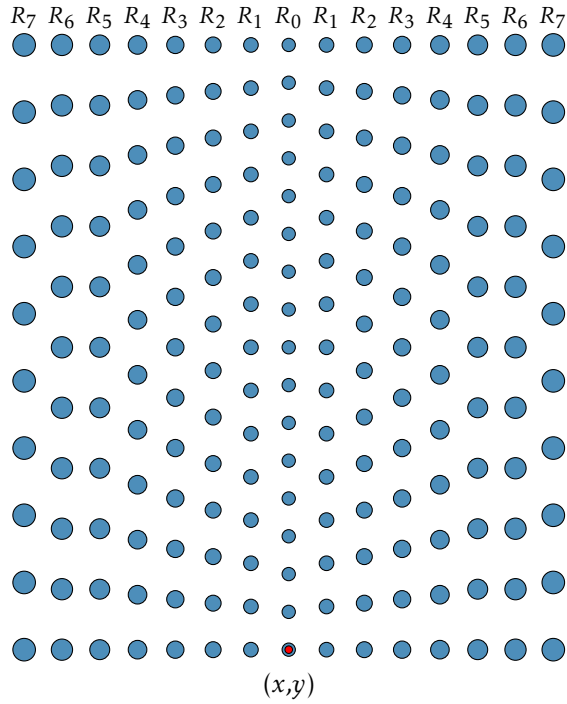


Figure 91: Double ramped array of shapes with a linearly varying vertical pitch and radius, and a constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the horizontal extent is S_A .

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p Δ S_A NUALinA4D>

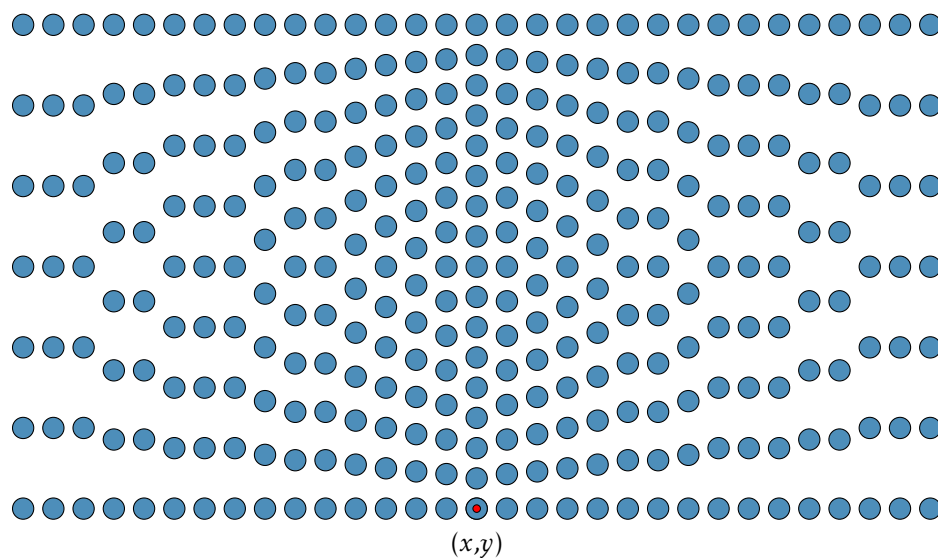


Figure 92: Double ramped array of shapes with a linearly varying vertical and constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the horizontal extent is $2S_A$.

<uniqueStructPrefix x y r_x r_y δ_r θ_s N_{sides} p Δ S_A NUALinA4DR>

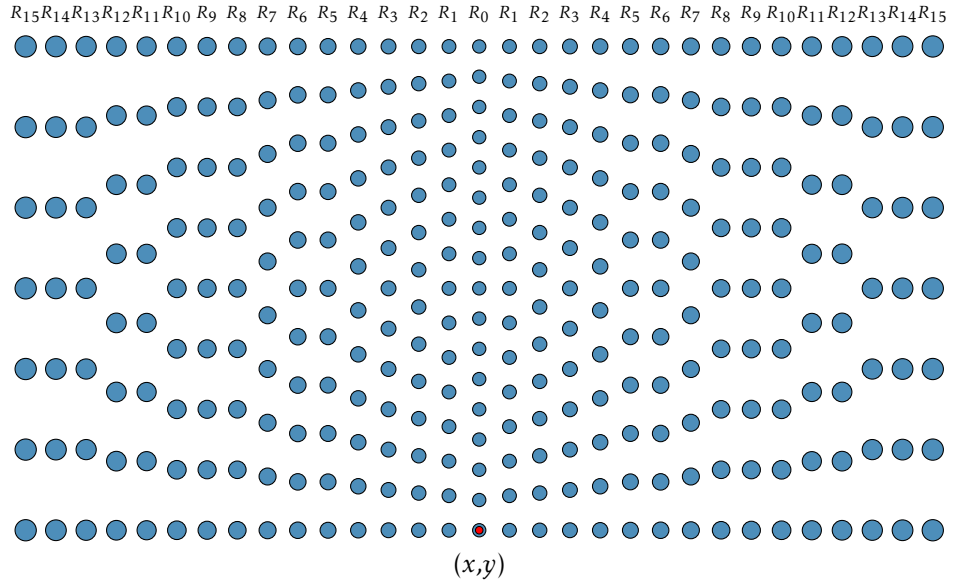


Figure 93: Double ramped array of shapes with a linearly varying vertical and constant horizontal pitch. The pitch of array elements along the y direction is scaled so that the array aligns with the horizontal extent is $2S_A$.

The above figure shows an array with a linearly varying shape radius across columns. The linearly varying R_i represents $r_x + i\delta_r$ and $r_y + i\delta_r$.

0.4.9.10 Exponentially varying pitch array

The non-uniform array shown below in figure 94 has an exponentially varying pitch between each column. The pitch increases exponentially between the values of p_{start} and p_{end} for $N_{columns}$ number of columns.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ NUAExp>

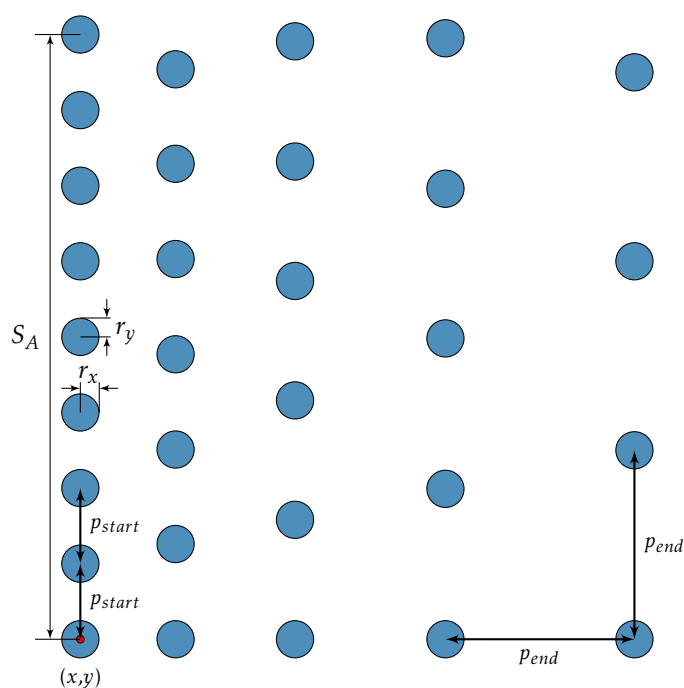


Figure 94: Array of shapes with an exponentially varying pitch between the pitch start (p_{start}) and end (p_{end}) values.

0.4.9.11 Exponentially varying pitch - aligned

Similar to the non-uniform array shown in section 0.4.9.10, the constructor **NUAExpA** creates an array where pitch between each columnar element is adjusted so that the end element coincides with the columnar extent S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ **NUAExpA**>

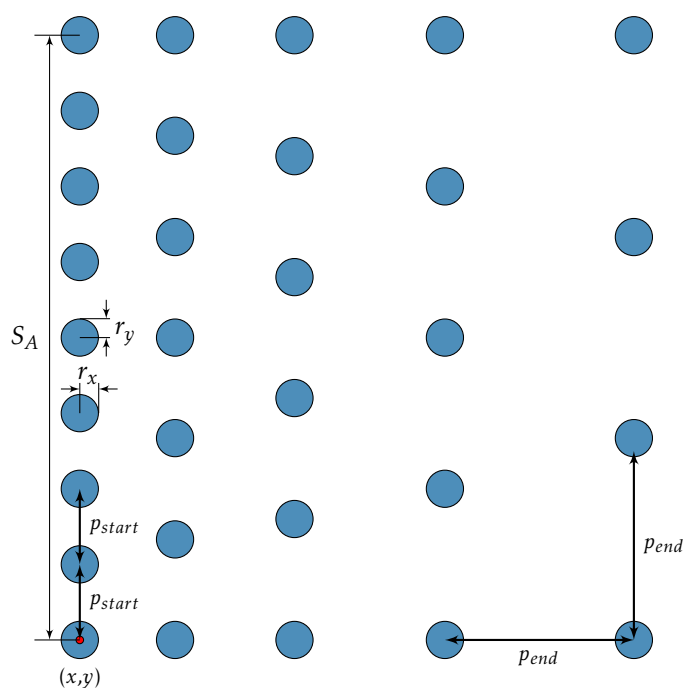


Figure 95: Array of shapes with an exponentially varying pitch between the pitch start (p_{start}) and end (p_{end}) values. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

0.4.9.12 Exponentially varying pitch symmetric array

Similar to the exponentially varying pitch array shown in the previous section 0.4.9.10, the below ramped array is composed of two exponentially varying pitch arrays. The total number of columns is $2N_{columns}$.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ NUAExp2>

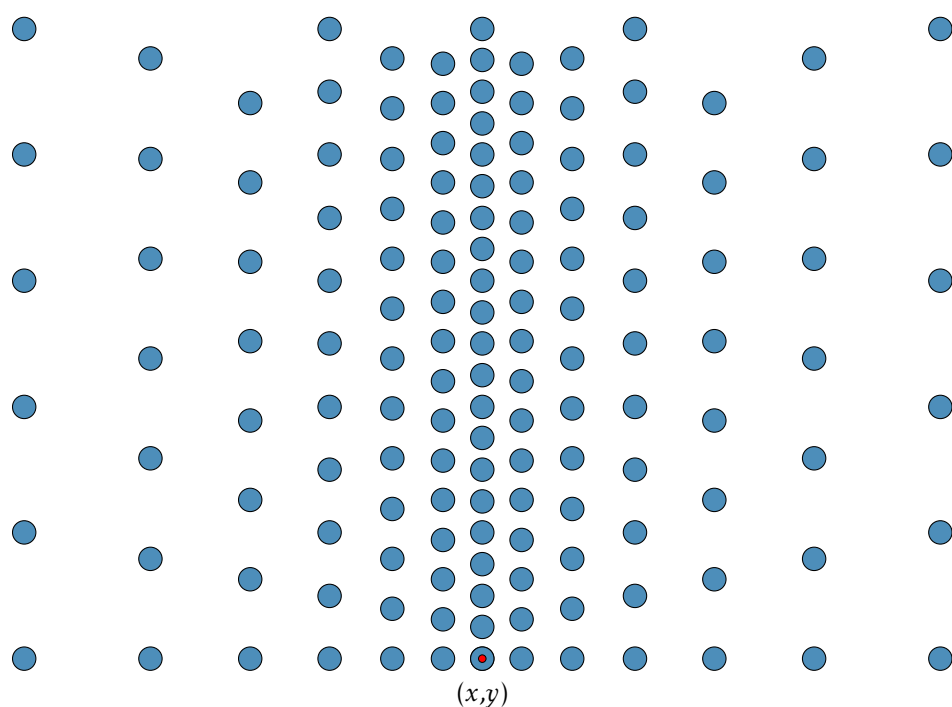


Figure 96: Ramped array of shapes with an exponentially varying pitch between the pitch start (p_{start}) and end (p_{end}) values.

0.4.9.13 Exponentially varying pitch, aligned symmetric array

Similar to the aligned exponentially varying pitch array shown in the previous section 0.4.9.11, the below ramped array is composed of two aligned exponentially varying pitch arrays. The total number of columns is $2N_{columns}$.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ NUAExpA2>

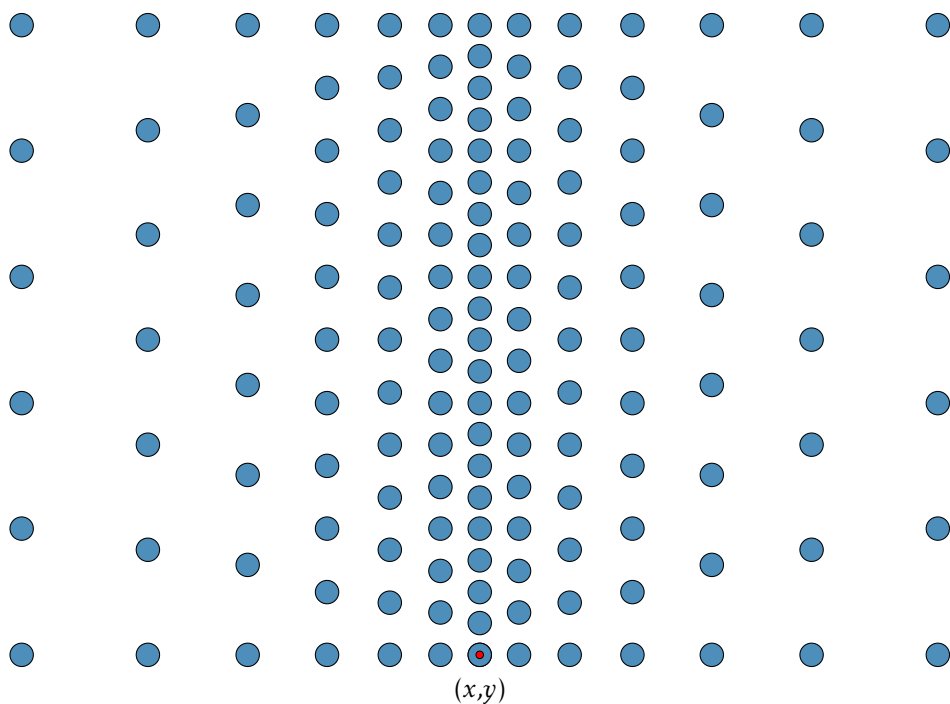


Figure 97: Ramped array of shapes with an exponentially varying pitch between the pitch start (p_{start}) and end (p_{end}) values. The pitch of array elements along the y direction is scaled so that the array aligns with the array size S_A .

0.4.9.14 Exponentially varying vertical and constant horizontal pitch array

The **NUAExp3** constructor create exponentially varying vertical and constant (p_{start}) horizontal pitch array.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ **NUAExp3**>

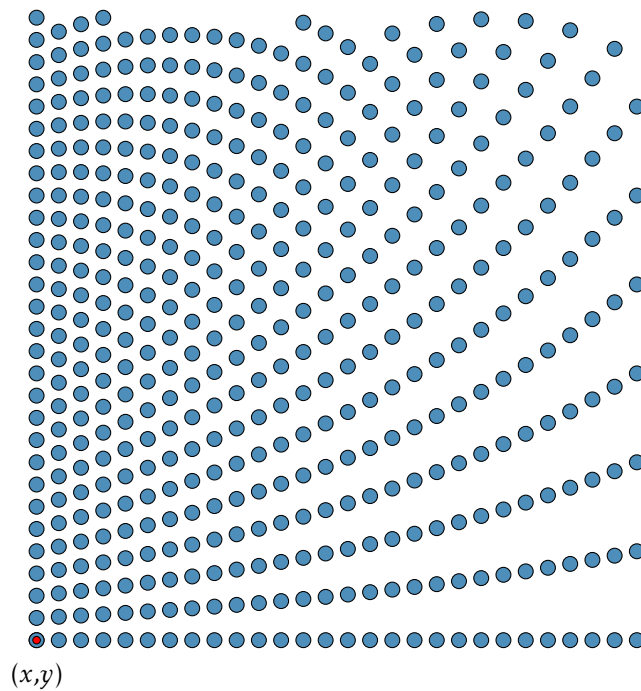


Figure 98: Exponentially varying vertical and constant horizontal pitch between the arrayed shapes.

0.4.9.15 Exponentially varying vertical and constant horizontal pitch, aligned array

Similar to the non-uniform array shown in the previous section 0.4.9.14, the constructor **NUAExpA3** creates an array where pitch between each row element is adjusted so that the end element coincides with the columnar extent S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ **NUAExpA3**>

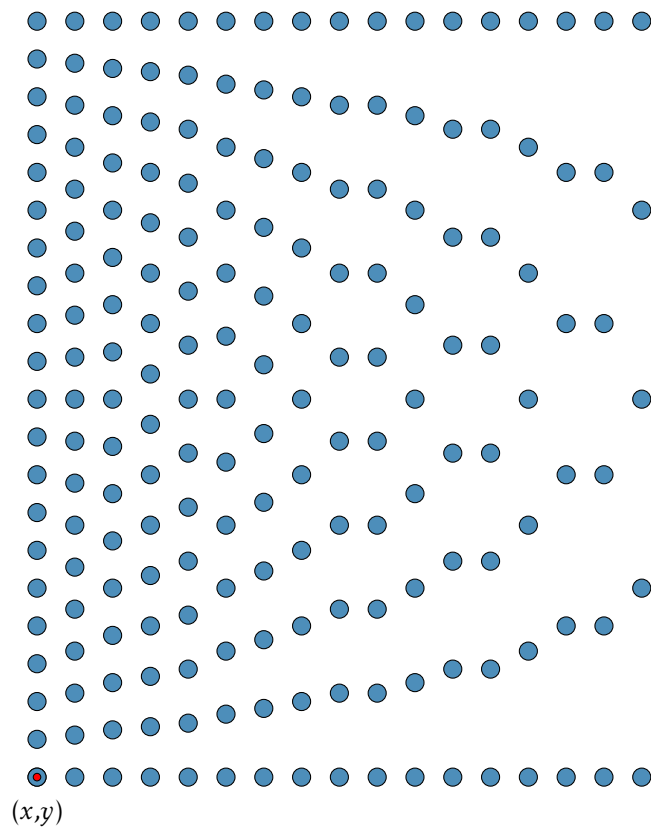


Figure 99: Exponentially varying vertical and constant horizontal pitch between the arrayed shapes.

0.4.9.16 Exponentially varying vertical and constant horizontal pitch symmetric array

Similar to the non-uniform array shown in figure 98, the constructor `NUAExp4` creates a symmetric array with elements of exponentially varying vertical and constant (p_{start}) horizontal pitch.

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ `NUAExp4`>

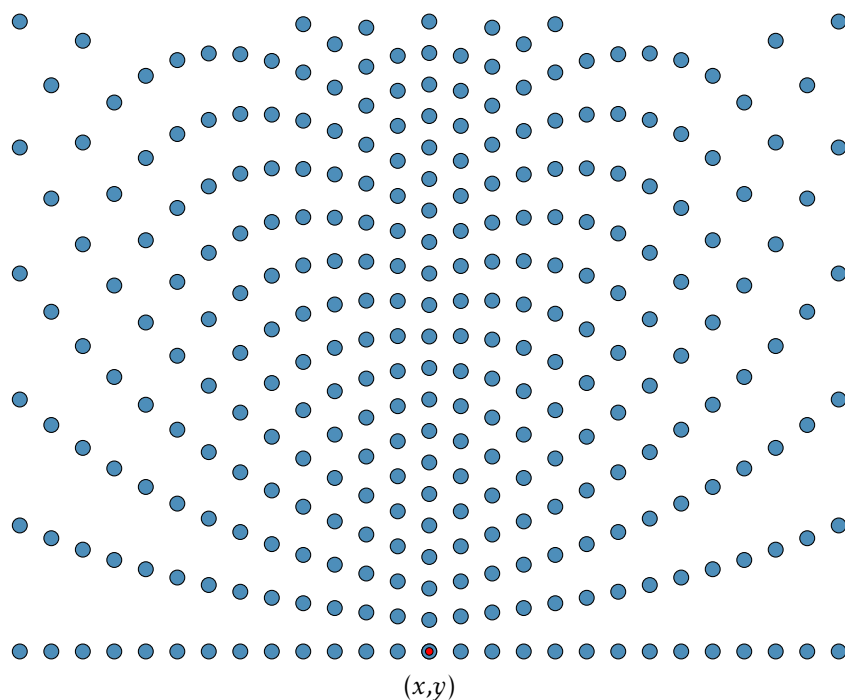


Figure 100: Exponentially varying vertical and constant horizontal pitch between the arrayed shapes. The array is symmetric about the y axis.

0.4.9.17 Exponentially varying vertical and constant horizontal pitch, symmetric aligned array

Similar to the non-uniform array shown in the previous section 0.4.9.16, the constructor **NUAExpA4** creates an array where pitch between each row element is adjusted so that the end element coincides with the columnar extent S_A .

<uniqueStructPrefix x y r_x r_y θ_s N_{sides} p_{start} p_{end} S_A $N_{columns}$ **NUAExpA4**>

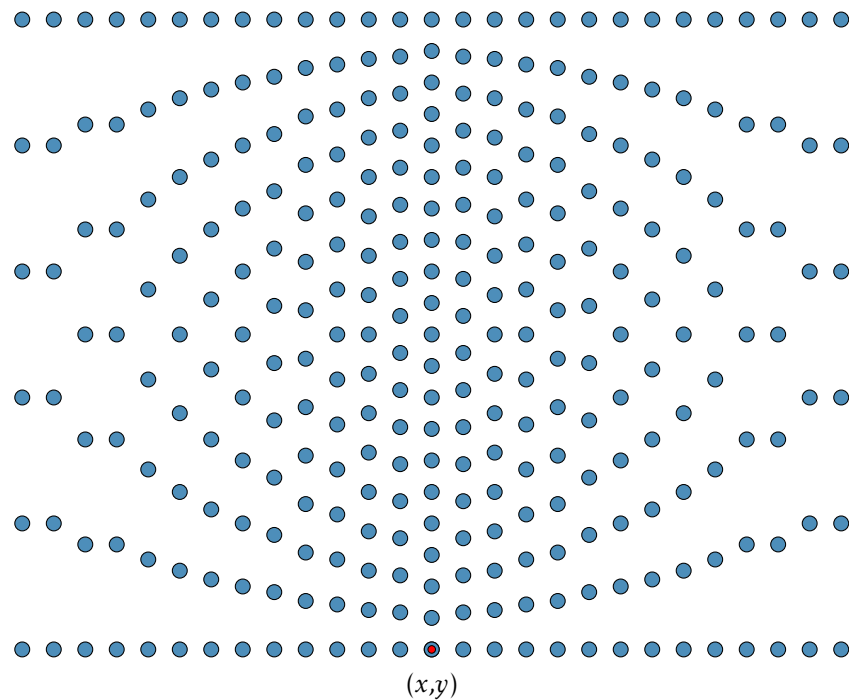


Figure 101: Exponentially varying vertical and constant horizontal pitch between the arrayed shapes. The array is symmetric about the y axis.

0.4.10 Paths and Shapes With Splines

The previously introduced constructor **polypath** connects coordinate points $(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$ sequentially, as seen in Figure 102a. The three constructors introduced here utilize spline interpolation between user defined (x, y) point pairs (Figure 102b). These methods have a constraint requiring that consecutive x coordinate values not be equal, i.e. $x_{i-1} \neq x_n \neq x_{i+1}$.

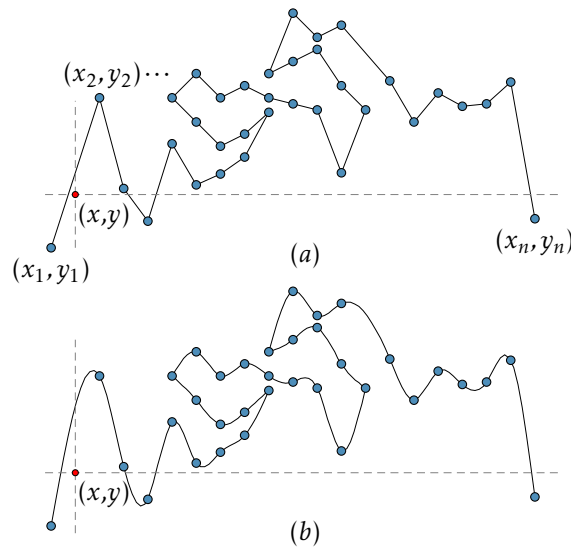


Figure 102: (a) straight line and (b) spline interpolated connections between sequential (x_i, y_i) coordinate points.

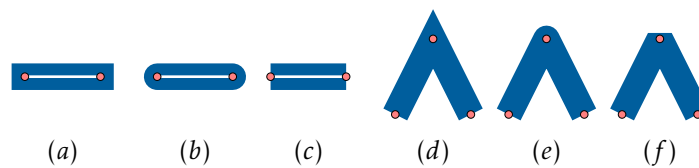


Figure 103: Schematic illustration of Cap and Join styles. (a) $\text{Cap(Butt)} = 0$, segment is cut at the endpoints. (b) $\text{Cap(round)} = 1$ circular cap centered on the endpoints with the diameter of the line width. (c) $\text{Cap(Square)} = 2$ square cap that extends by half of the line width. (d) $\text{Join(Miter)} = 0$ joint that extends the outside line edges until they meet. (e) $\text{Join(Round)} = 1$ joint that rounds off corners with circle diameter equal to the line width. (f) $\text{Join(Bevel)} = 2$ joint that connects outside line corners with a straight line.

The resulting spline interpolated structures are smooth curves with sharp edges at the turning coordinates points where x changes direction. The turning points can be smoothed out by choosing rounded joining segments (Join= 1 Figure 103e). The three constructors are composed of (x, y) point pairs (dots in Figures 102, 104 and 105), N_p are the number of spline interpolation points, two spline paths have W representing path width, Cap and Join parameters are defined in Figure 103, and F_N is the number of fractured segments. Figure 102b shows a closed spline path that's fractured into 4 slices ($F_N = 4$). Paths with large number of vertices can exceed the value allowed by the GDS standard. Fracturing shapes into several shapes allows users to create adjoining shapes with less vertices. Fracturing occurs along the y direction.

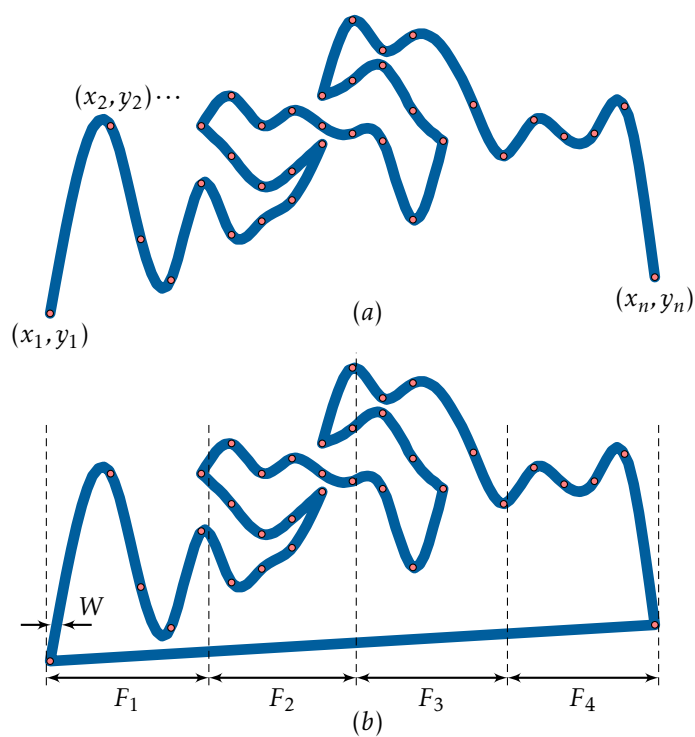


Figure 104: Spline path constructors (a) `splinePath` and (b) `splineClosedPath` with adjoining fractured segments ($F_N = 4$). The closed path connects the last (x_n, y_n) and first (x_1, y_1) points.

x_1 y_1 x_2 y_2 \cdots x_n y_n N_p W Cap $Join$ F_N `splinePath`

x_1 y_1 x_2 y_2 \cdots x_n y_n N_p W Cap $Join$ F_N `splineClosedPath`

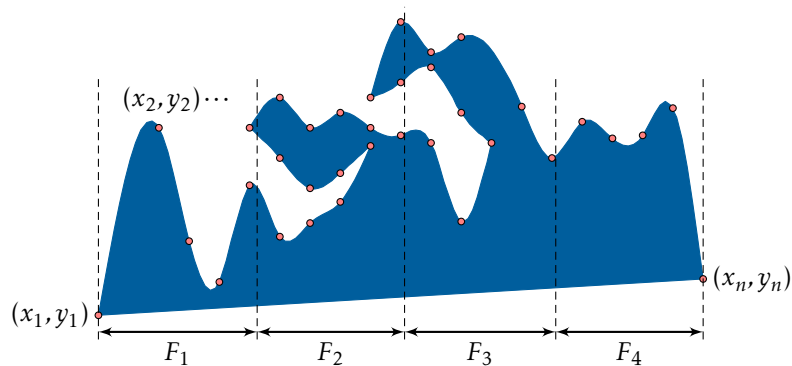


Figure 105: Spline shape constructor `splineShape`. The shape is fractured into adjoining shape segments with parameter $F_N = 4$. The resulting shape is constructed by connections between the spline interpolated points (N_p) and the connection between the last (x_n, y_n) and first (x_1, y_1) points.

x_1 y_1 x_2 y_2 \cdots x_n y_n N_p *Cap* *Join* F_N `splineShape`

0.4.11 Resolution Test Pattern

L-shaped line space resolution pattern defined by a line width w , lengths L_1 and L_2 , pitches p_1 and p_2 , and number of lines N .

x y w L_1 L_2 p_1 p_2 N `resoPatternLS2`

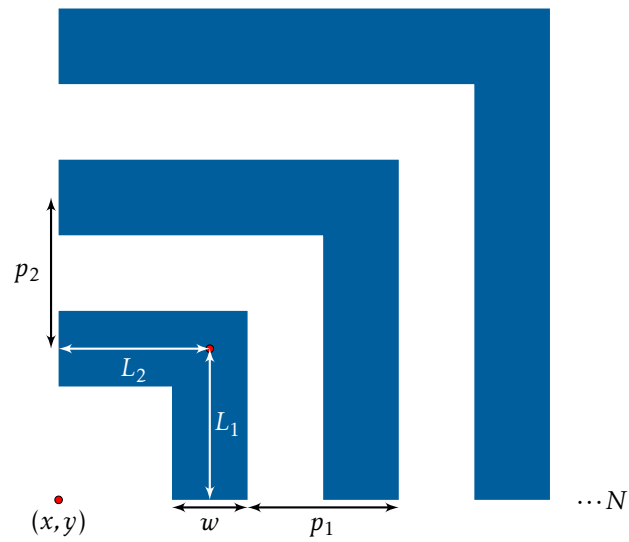


Figure 106: Resolution pattern.

0.4.12 Rounded Paths and Shapes

The following methods create rounded corners along a path or shape. Paths of width w can either be open or closed. The curved segments are defined by a radius r and the number of vertices N . The F_N parameter defined the number of segments the resulting GDS shape is fractured into (see Fig. 104). Paths with large number of vertices can exceed the value allowed by the GDS standard. Fracturing shapes into several shapes allows users to create adjoining shapes with less vertices. Fracturing occurs along the y direction.

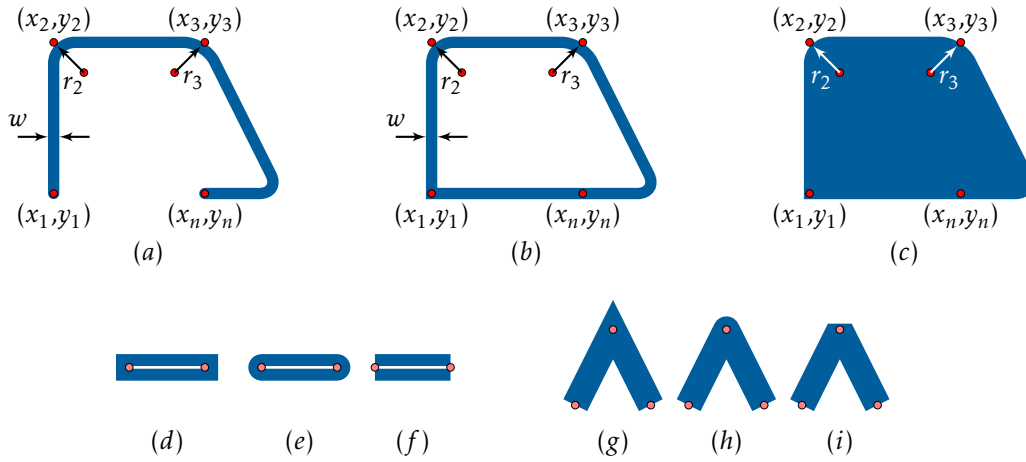


Figure 107: Rounded (a) open path (b) closed path (c) shape. (d) $Cap(Butt) = 0$, segment is cut at the endpoints. (e) $Cap(round) = 1$ circular cap centered on the endpoints with the diameter of the line width. (f) $Cap(Square) = 2$ square cap that extends by half of the line width. (g) $Join(Miter) = 0$ joint that extends the outside line edges until they meet. (h) $Join(Round) = 1$ joint that rounds off corners with circle diameter equal to the line width. (i) $Join(Bevel) = 2$ joint that connects outside line corners with a straight line.

$x_1 \ y_1 \ x_2 \ y_2 \ r_2 \ N_2 \ x_3 \ y_3 \ r_3 \ N_3 \ x_4 \ y_4 \ r_4 \ N_4 \ \dots \ x_n \ y_n \ w \ Cap \ Join \ F_N \ \text{roundPath}$

$x_1 \ y_1 \ x_2 \ y_2 \ r_2 \ N_2 \ x_3 \ y_3 \ r_3 \ N_3 \ x_4 \ y_4 \ r_4 \ N_4 \ \dots \ x_n \ y_n \ w \ Cap \ Join \ F_N \ \text{roundClosedPath}$

$x_1 \ y_1 \ x_2 \ y_2 \ r_2 \ N_2 \ x_3 \ y_3 \ r_3 \ N_3 \ x_4 \ y_4 \ r_4 \ N_4 \ \dots \ x_n \ y_n \ \ Cap \ Join \ F_N \ \text{roundShape}$

0.4.13 Spin-Ice Lattice

Artificial spin-ice lattice.

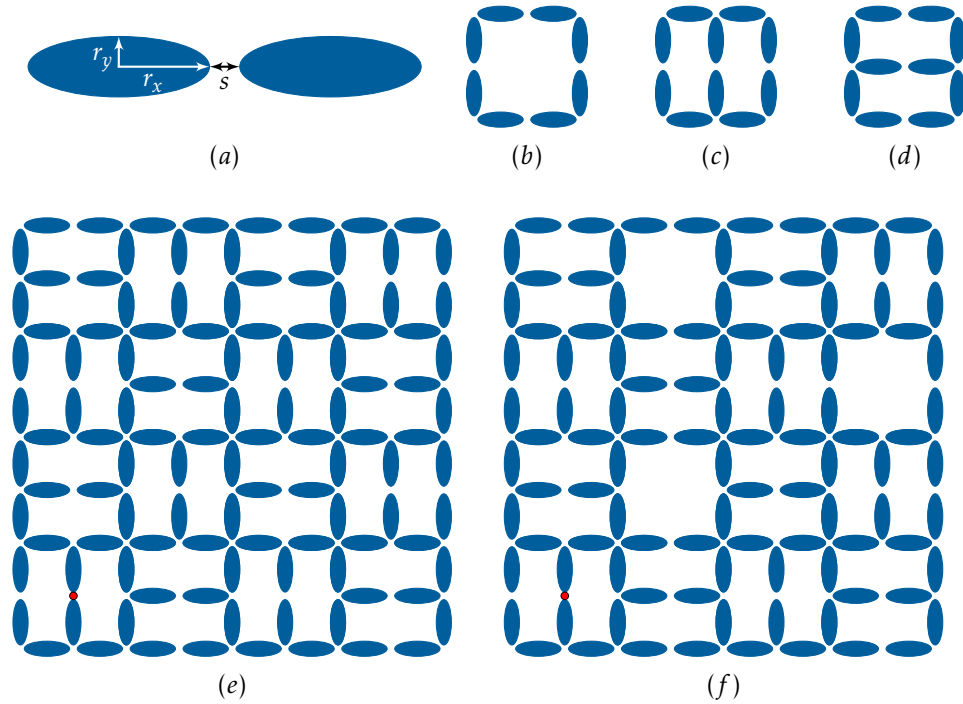


Figure 108: Artificial spin-ice structure. (a) Single cell containing two ellipses. (b) Empty, (c) vertical and (d) horizontal cells. (e) Array composed of vertical and horizontal cells. (f) Array with a percentage of empty cells.

```
<uniqueStructName x y r_x r_y s P_e N_sides N_x N_y spinIceA>
```

```
<uniqueStructName x y r_x r_y s P_e N_x N_y spinIceVectorA>
```

These constructors create ellipses with N_{sides} or vectorized ellipses defined by [shapeReso](#). Suffixes *E*, *H* and *V* are appended to the *uniqueStructName* string to define structures that hold the empty, horizontal and vertical cells respectively. N_x and N_y define the number of elements in the x (columns) and y (rows) directions. P_e is the percentage of empty cells within the $N_x \times N_y$ array, $0 \leq P_e \leq 100$.

The following constructor creates patterns identical to ones defined in Figure 108 with a rounded rectangle as the base element. The rendering resolution of the rounded rectangle is defined by the `shapeReso` parameter. The radii of the rounded rectangle are defined as $r_x = r_y = H/2$.

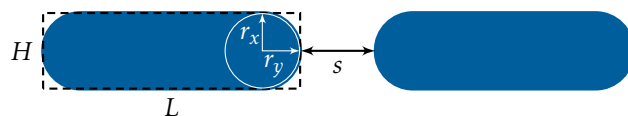


Figure 109: Rounded rectangle single cell used to construct the artificial spin ice structure showing in Figure 108.

`<uniqueStructName x y L H s Pe Nx Ny spinIceB>`

0.4.14 Van der Pauw Structures

Van der Pauw structures created using GDS layers L_a and L_b .

x y d w L_1 L_2 a b L_a L_b **vanDerPauwV1**

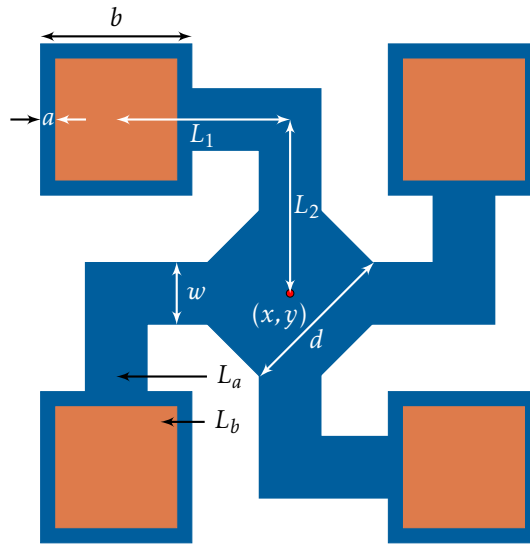


Figure 110: 4 terminal **vanDerPauwV1** structure.

x y d w L_1 L_2 L_3 L_4 a b L_a L_b **vanDerPauwV2**

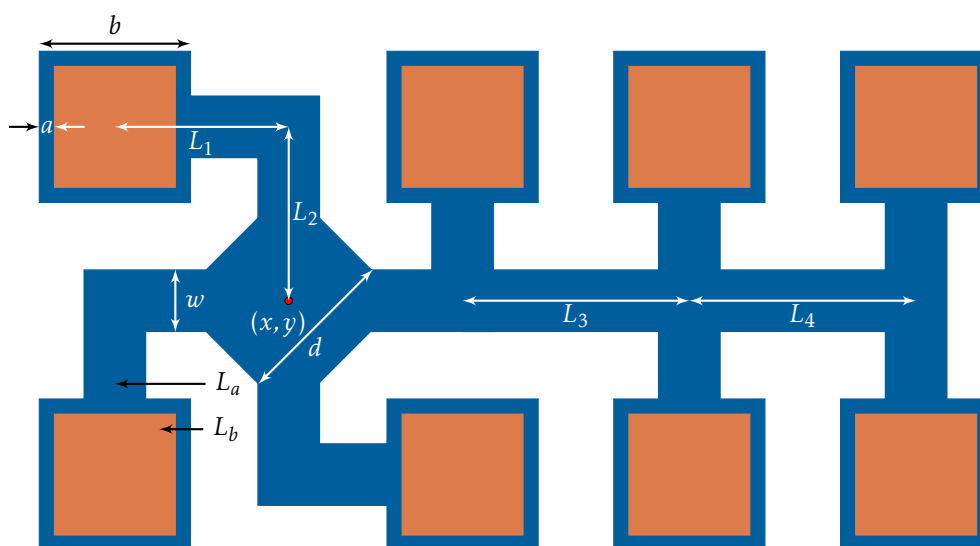


Figure 111: 8 terminal **vanDerPauwV2** structure.

0.4.15 Wire Contacts

x y w h_1 h_2 h_3 L_a L_b L_c **wireC1**

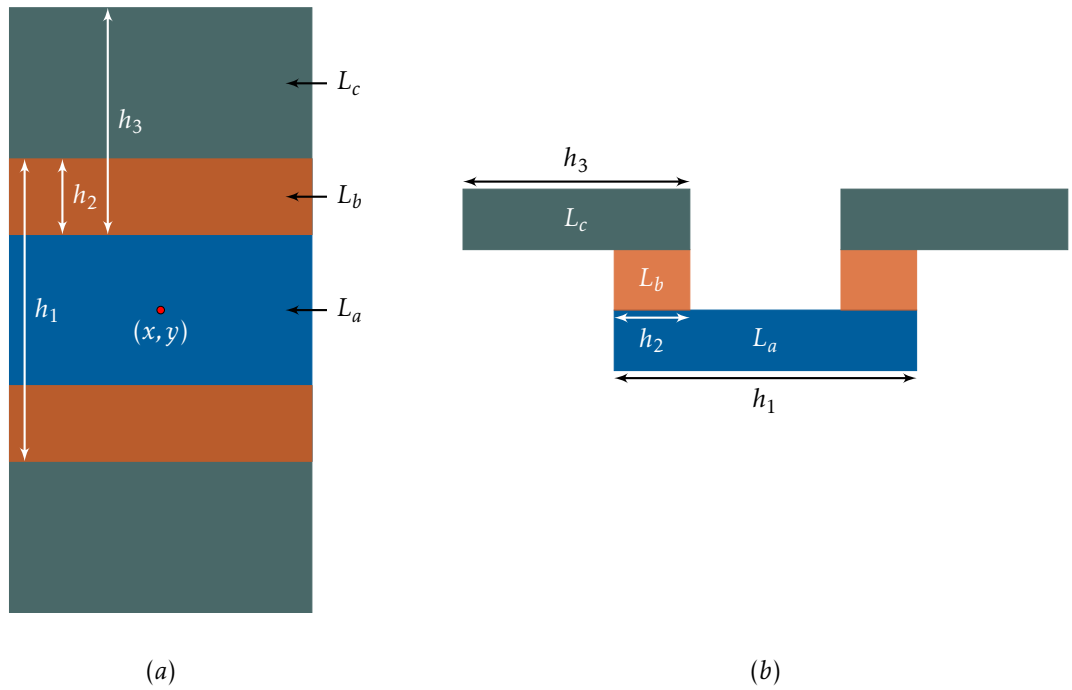


Figure 112: (a) Top-down and (b) cross-section views of the **wireC1** structure.

0.5 Shapes - CNST Special Scripts

0.5.1 Circles and Ellipses

0.5.1.1 Primitive Circle and Ellipse

Replace the 2016 manual release with this page. The shapes are centered at (x, y) , defined by either a circular radius r or elliptical radii r_x and r_y , the number of sides (N_{sides}) and a rotation 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 y r N_{sides} **circle**

x y r_x r_y N_{sides} $\theta_{(x,y)}$ **ellipse**

0.5.1.2 Vectorized Circle and Ellipse

The shapes are constructed from Bezier curves, centered at (x, y) , defined by either a circular radius r or elliptical radii r_x and r_y , and a rotation 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 **shapeReso** parameter (see section 0.9.4). 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 y r **circleVector**

x y r_x r_y $\theta_{(x,y)}$ **ellipseVector**

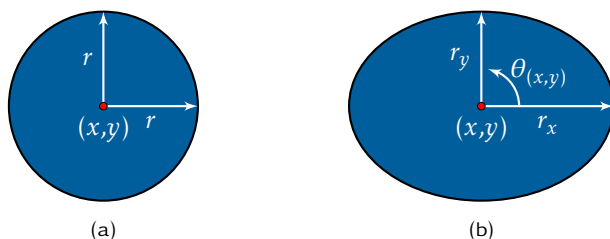


Figure 113: (a) Circle and (b) ellipse shapes.

Below constructors for outlined circle and ellipse shapes. The defining shape parameters are identical to ones described in the previous section with the addition of an outline width w_o .

0.5.1.3 Primitive Circle and Ellipse Outline

$x \ y \ r \ w_o \ N_{sides}$ **circleOutline**

$x \ y \ r_x \ r_y \ w_o \ N_{sides} \ \theta_{(x,y)}$ **ellipseOutline**

0.5.1.4 Vectorized Circle and Ellipse Outline

$x \ y \ r \ w_o$ **circleOutlineVector**

$x \ y \ r_x \ r_y \ w_o \ \theta_{(x,y)}$ **ellipseOutlineVector**

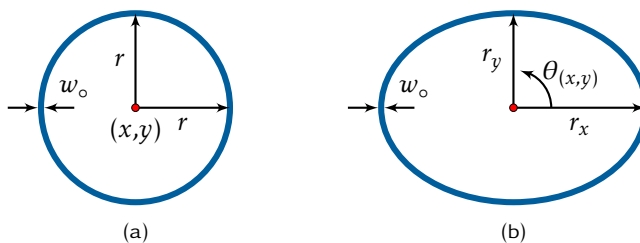


Figure 114: Outline (a) circle and (b) ellipse shapes.

Below constructors for dashed circle and ellipse shapes. The defining shape parameters are identical to ones described in the previous section with the addition of a dash length and gap defined by d_L and d_G , respectively. The *Cap* and *Join*, shown in Figure 116, define the end-cap and line-join parameters.

0.5.1.5 Primitive Circle and Ellipse Dashed

$x \ y \ r \ w_o \ N_{sides} \ d_L \ d_G \ Cap \ Join$ **circleDashed**

$x \ y \ r_x \ r_y \ w_o \ N_{sides} \ d_L \ d_G \ Cap \ Join \ \theta_{(x,y)}$ **ellipseDashed**

0.5.1.6 Vectorized Circle and Ellipse Dashed

$x \ y \ r \ w_o \ d_L \ d_G \ Cap \ Join$ **circleDashedVector**

$x \ y \ r_x \ r_y \ w_o \ d_L \ d_G \ Cap \ Join \ \theta_{(x,y)}$ **ellipseDashedVector**

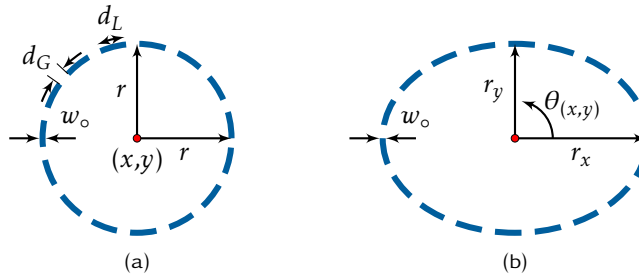


Figure 115: Dashed (a) circle and (b) ellipse shapes.

0.5.2 Dashed Structures - Cap and Join

0.5.2.1 End-Cap and Line-Join Parameters

The end-cap and line-join parameters are used to define the lines within dashed structures. Figure 116 shows the Cap and Join parameters for shapes that are defined by dashed lines.

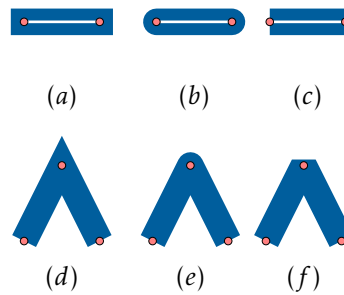


Figure 116: (a) $Cap(Butt) = 0$, segment is cut at the endpoints. (b) $Cap(round) = 1$ circular cap centered on the endpoints with the diameter of the line width. (c) $Cap(Square) = 2$ square cap that extends by half of the line width. (d) $Join(Miter) = 0$ joint that extends the outside line edges until they meet. (e) $Join(Round) = 1$ joint that rounds off corners with circle diameter equal to the line width. (f) $Join(Bevel) = 2$ joint that connects outside line corners with a straight line.

0.5.3 Cross Shapes

0.5.3.1 Cross

Replace the 2016 manual release with this page. The cross shape 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.

x y W L $\theta_{(x,y)}$ `cross`

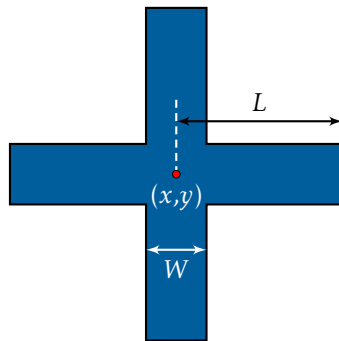


Figure 117: `cross` constructor parameters.

0.5.3.2 Cross - Outline and Dashed

The cross shape outline 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. The outline width is defined by w_o . Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

$x \ y \ W \ L \ w_o \ \theta_{(x,y)}$ **crossOutline**

$x \ y \ W \ L \ w_o \ d_L \ d_G \ Cap \ Join \ \theta_{(x,y)}$ **crossDashed**

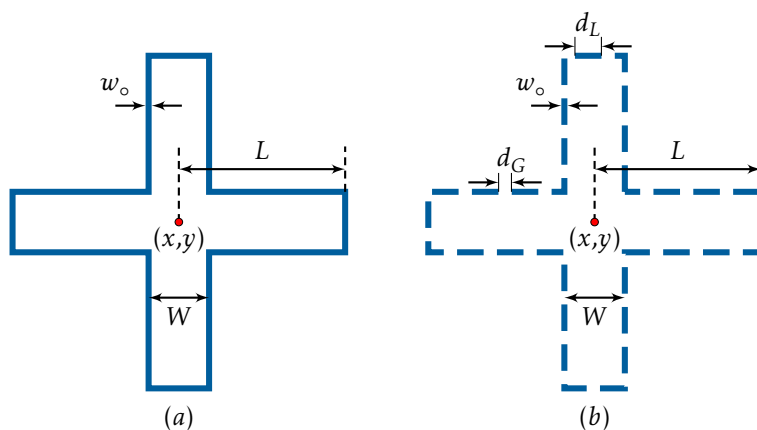


Figure 118: (a) **crossOutline** and (b) **crossDashed** constructor parameters.

0.5.3.3 Parameterized Cross

The cross below 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.

x y W_1 L_1 W_2 L_2 $\theta_{(x,y)}$ `crossV2`

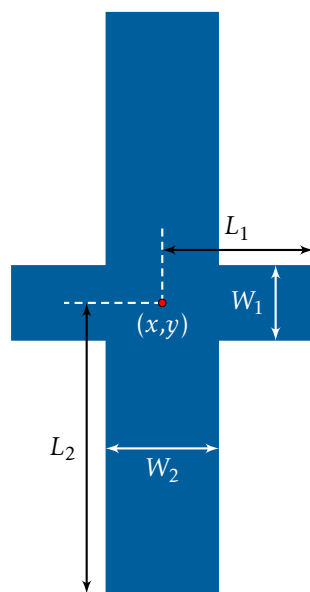


Figure 119: `crossV2` constructor parameters.

0.5.3.4 Parameterized Cross - Outline and Dashed

The outline and dashed crosses illustrated below are defined by the center (x, y) , widths W_1 and W_2 , lengths L_1 and L_2 , line width w_o and rotation about the center at an angle $\theta_{(x,y)}$ expressed in degrees. Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x y W_1 L_1 W_2 L_2 w_o $\theta_{(x,y)}$ **crossOutlineV2**

x y W_1 L_1 W_2 L_2 w_o d_L d_G *Cap* *Join* $\theta_{(x,y)}$ **crossDashedV2**

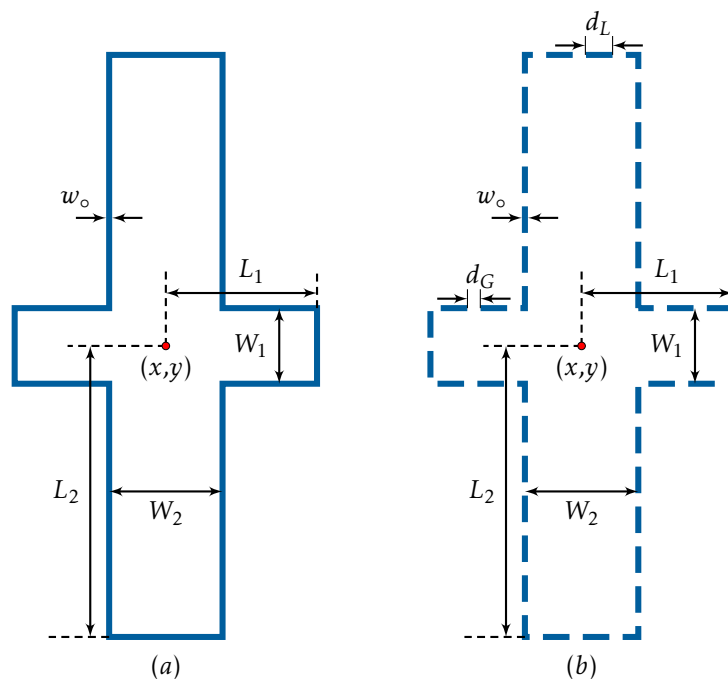


Figure 120: (a) **crossOutlineV2** and (b) **crossDashedV2** constructor parameters.

0.5.3.5 Rounded Cross

The rounded cross below is defined by the center (x, y) , width W , length L , radius r and rotation about the center at an angle $\theta_{(x,y)}$ expressed in degrees.

x y W L r $\theta_{(x,y)}$ `crossRound`

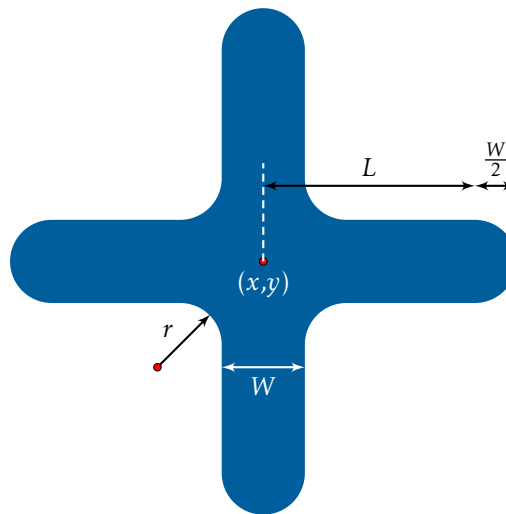


Figure 121: `crossRound` constructor parameters.

0.5.3.6 Rounded Cross - Outline and Dashed

The rounded crosses shown below are defined by the center (x, y) , width W , length L , radius r , line width w_o and rotation about the center at an angle $\theta_{(x,y)}$ expressed in degrees. Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x y W L r w_o $\theta_{(x,y)}$ **crossRoundOutline**

x y W L r w_o d_L d_G *Cap* *Join* $\theta_{(x,y)}$ **crossRoundDashed**

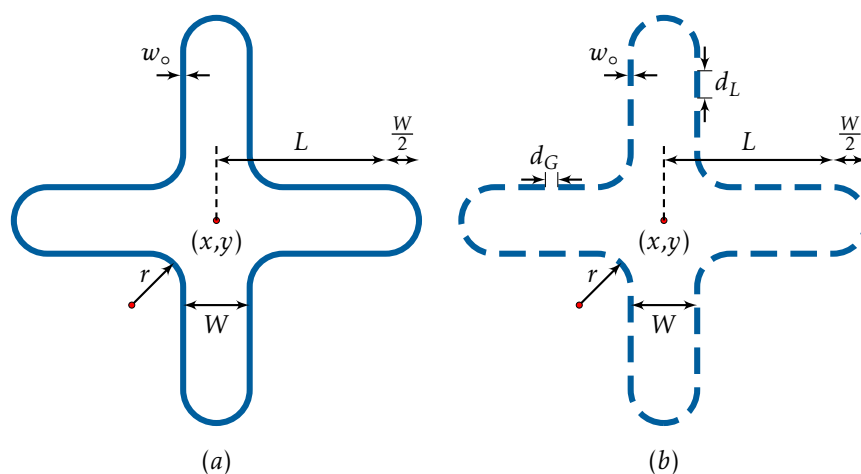


Figure 122: (a) **crossRoundOutline** and (b) **crossRoundDashed** constructor parameters.

0.5.3.7 Parameterized Rounded Cross

The rounded cross below is defined by the center (x,y) , widths W_1 and W_2 , lengths L_1 and L_2 , radius r and rotation about the center at an angle $\theta_{(x,y)}$ expressed in degrees.

x y W_1 L_1 W_2 L_2 r $\theta_{(x,y)}$ `crossRoundV2`

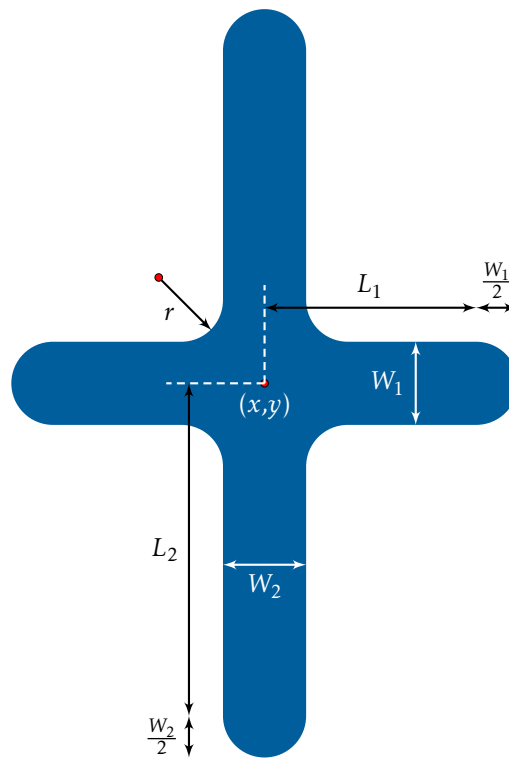


Figure 123: `crossRoundV2` constructor parameters.

0.5.3.8 Parameterized Rounded Cross - Outline and Dashed

The rounded crosses shown below are defined by the center (x, y) , widths W_1 and W_2 , lengths L_1 and L_2 , radius r , line width w_o and rotation about the center at an angle $\theta_{(x,y)}$ expressed in degrees. Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x y W_1 L_1 W_2 L_2 r w_o $\theta_{(x,y)}$ **crossRoundOutlineV2**

x y W_1 L_1 W_2 L_2 r w_o d_L d_G *Cap* *Join* $\theta_{(x,y)}$ **crossRoundDashedV2**

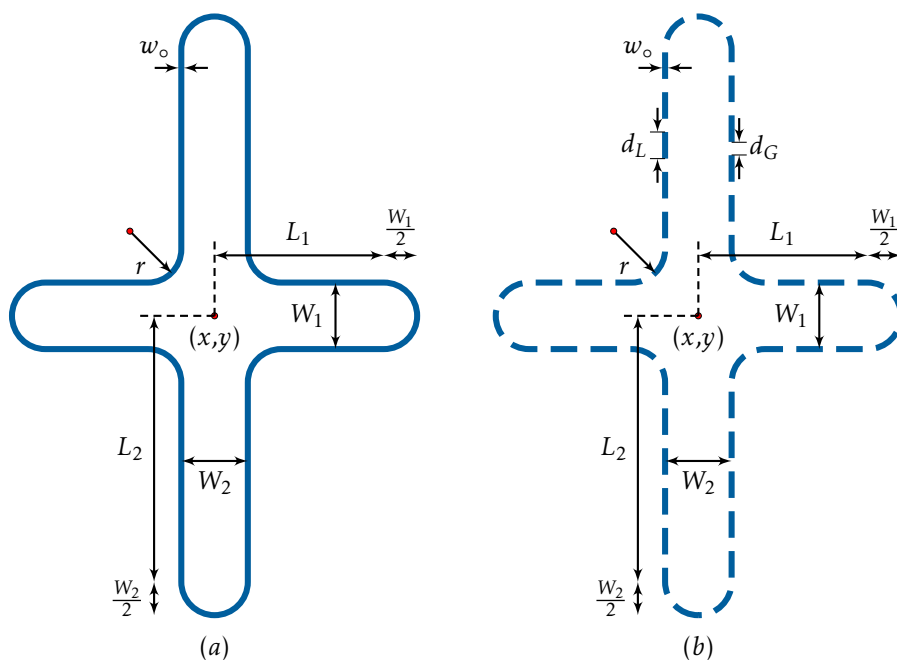


Figure 124: **crossRoundV2** constructor parameters.

0.5.4 L-Shapes

0.5.4.1 L-Shape

Replace the 2016 manual release with this page. 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.

x y W_1 L_1 W_2 L_2 $\theta_{(x,y)}$ `Lshape`

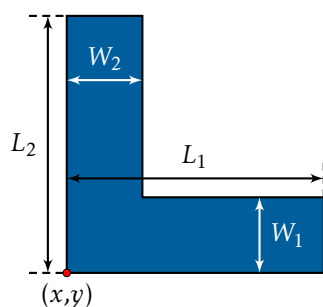


Figure 125: `Lshape` constructor parameters.

0.5.4.2 L-Shape - Outline and Dashed

L-shape is defined by the center (x, y) , widths (W_1 and W_2), lengths (L_1 and L_2), line width w_o and rotation about the center at an angle $\theta_{(x,y)}$ expressed in degrees. Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x y W_1 L_1 W_2 L_2 w_o $\theta_{(x,y)}$ **LshapeOutline**

x y W_1 L_1 W_2 L_2 w_o d_L d_G *Cap* *Join* $\theta_{(x,y)}$ **LshapeDashed**

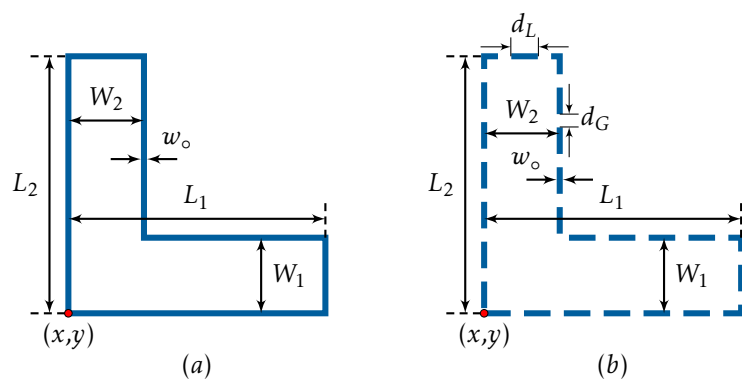


Figure 126: (a) **LshapeOutline** and (b) **LshapeDashed** constructor parameters.

0.5.4.3 L-Shape Round

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

x y W_1 L_1 W_2 L_2 r $\theta_{(x,y)}$ **LshapeRound**

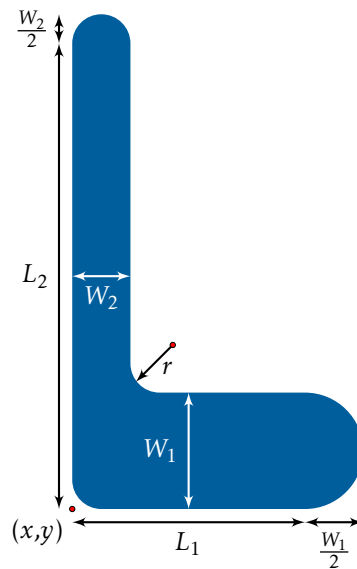


Figure 127: **LshapeRound** constructor parameters.

0.5.4.4 L-Shape Round - Outline and Dashed

L-shape is defined by the center (x, y) , widths (W_1 and W_2), lengths (L_1 and L_2), radius r between the two length segments, line width w_o and rotation about the center at an angle $\theta_{(x,y)}$ expressed in degrees. Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x y W_1 L_1 W_2 L_2 r w_o $\theta_{(x,y)}$ **LshapeRoundOutline**

x y W_1 L_1 W_2 L_2 r w_o d_L d_G *Cap* *Join* $\theta_{(x,y)}$ **LshapeRoundDashed**

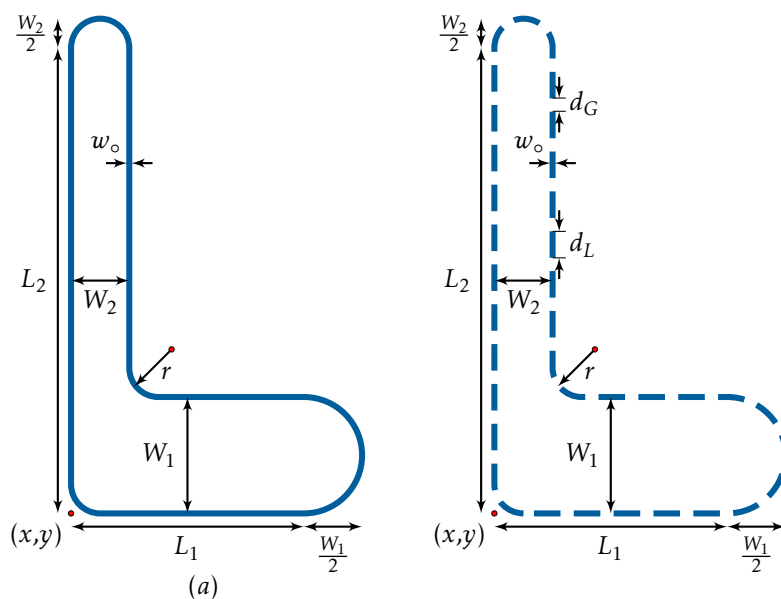


Figure 128: (a) **LshapeRoundOutline** and (b) **LshapeRoundDashed** constructor parameters.

0.5.5 Rectangular and Square Shapes

0.5.5.1 Rectangle

Replace the 2016 manual release with this page. The shape consists of four 90° corners. Three constructors are used to define the rectangular object.

x_{LL}	y_{LL}	x_{UR}	y_{UR}	$\theta_{(x_{LL}, y_{LL})}$	rectangle
x_{LL}	y_{LL}	L	H	$\theta_{(x_{LL}, y_{LL})}$	rectangleLH
x_C	y_C	L	H	$\theta_{(x_C, y_C)}$	rectangleC

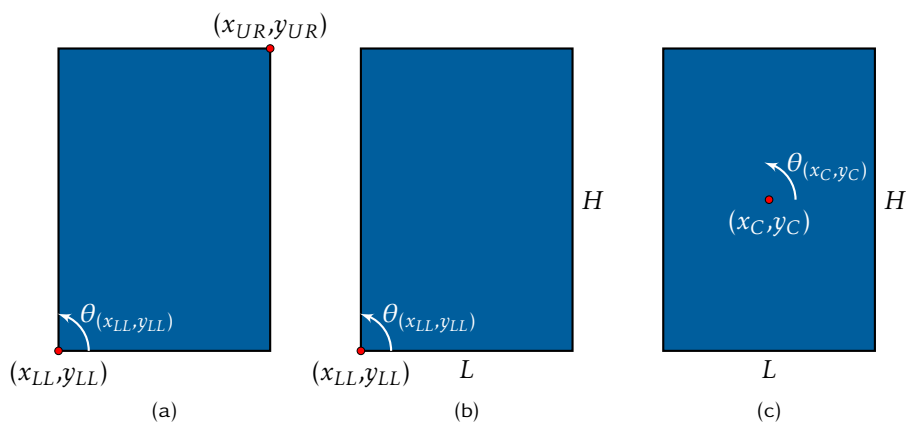


Figure 129: Three rectangle constructors. (a) **rectangle**, (b) **rectangleLH**, and (c) **rectangleC**.

0.5.5.2 Rectangle Outline

Below are three versions of the outline rectangle structures. The defining shape parameters are identical to ones described in the previous section with the addition of an outline width w_o .

x_{LL}	y_{LL}	x_{UR}	y_{UR}	w_o	$\theta_{(x_{LL},y_{LL})}$	<code>rectangleOutline</code>
x_{LL}	y_{LL}	L	H	w_o	$\theta_{(x_{LL},y_{LL})}$	<code>rectangleOutlineLH</code>
x_C	y_C	L	H	w_o	$\theta_{(x_C,y_C)}$	<code>rectangleOutlineC</code>

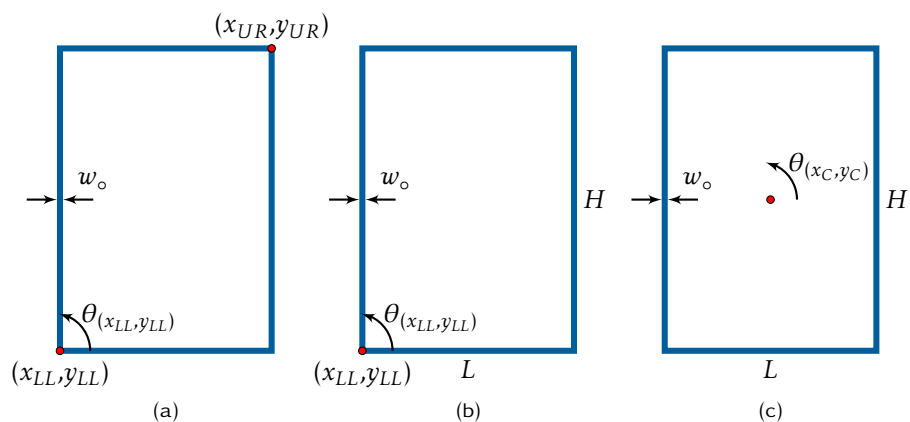


Figure 130: (a) `rectangleOutline`, (b) `rectangleOutlineLH`, and (c) `rectangleOutlineC` shapes.

0.5.5.3 Rectangle Dashed

Below are three versions of the dashed rectangle structures. The defining shape parameters are identical to ones described in the previous section with the addition of a dash length and gap defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x_{LL}	y_{LL}	x_{UR}	y_{UR}	w_o	d_L	d_G	Cap	Join	$\theta_{(x_{LL}, y_{LL})}$	rectangleDashed
x_{LL}	y_{LL}	L	H	w_o	d_L	d_G	Cap	Join	$\theta_{(x_{LL}, y_{LL})}$	rectangleDashedLH
x_C	y_C	L	H	w_o	d_L	d_G	Cap	Join	$\theta_{(x_C, y_C)}$	rectangleDashedC

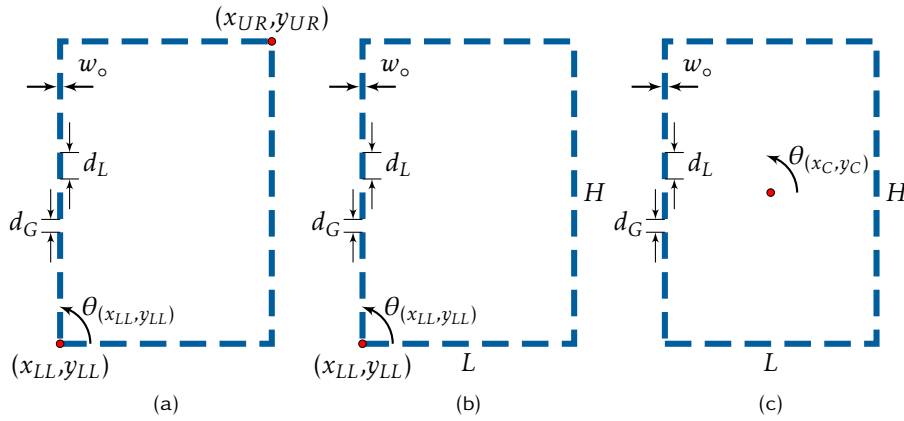


Figure 131: (a) **rectangleDashed**, (b) **rectangleDashedLH**, and (c) **rectangleDashedC** shapes.

0.5.5.4 Square

Below are two constructors for squares defined by the lower left and center coordinates.

$$\begin{array}{cccccc} x_{LL} & y_{LL} & L & \theta_{(x_{LL}, y_{LL})} & \text{square} \\ x_C & y_C & L & \theta_{(x_C, y_C)} & \text{squareC} \end{array}$$

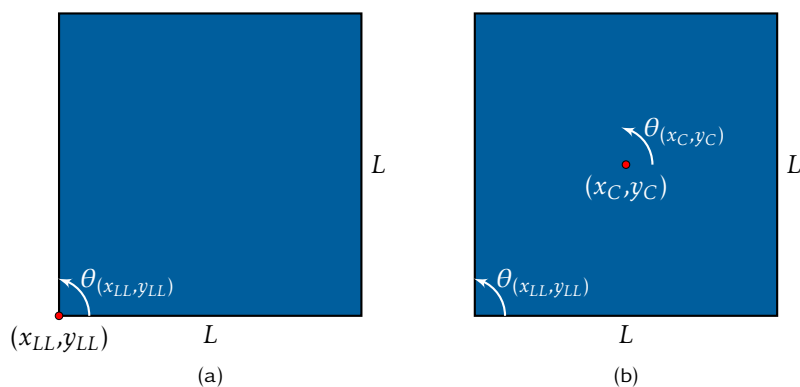


Figure 132: Three rectangle constructors. (a) *square* and (b) *squareC*.

0.5.5.5 Square Outline

Below are two versions of the outline square structures defined by the lower left and center coordinates. The defining shape parameters are identical to ones described in the previous section with the addition of an outline width w_o .

x_{LL} y_{LL} L w_o $\theta_{(x_{LL}, y_{LL})}$ **squareOutline**

x_C y_C L w_o $\theta_{(x_C, y_C)}$ **squareOutlineC**

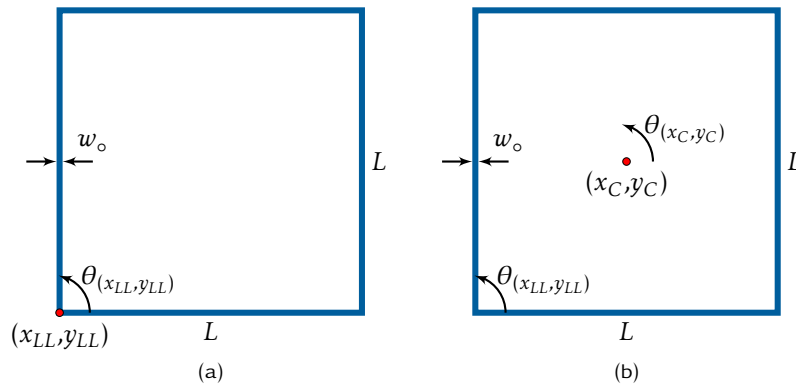


Figure 133: (a) **squareOutline** and (b) **squareOutlineC** shapes.

0.5.5.6 Square Dashed

Below are two versions of the dashed square structures defined by the lower left and center coordinates. The defining shape parameters are identical to ones described in the previous section with the addition of a dash length and gap defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x_{LL} y_{LL} L w_o d_L d_G Cap $Join$ $\theta_{(x_{LL}, y_{LL})}$ **squareDashed**

x_C y_C L w_o d_L d_G Cap $Join$ $\theta_{(x_C, y_C)}$ **squareDashedC**

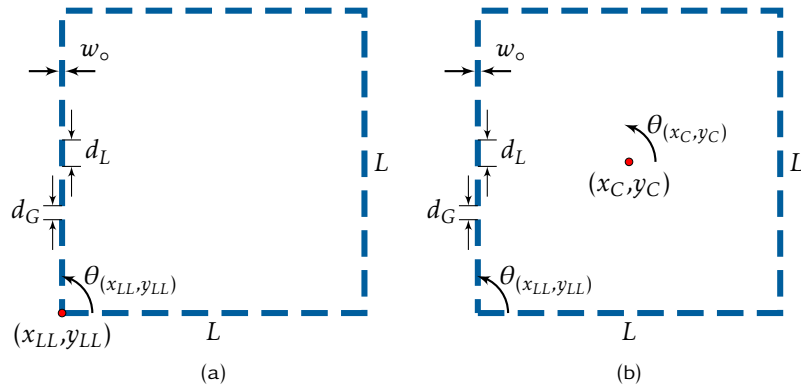


Figure 134: (a) **squareDashed** and (b) **squareDashedC** shapes.

0.5.5.7 Rounded Rectangle

Update 2016 release of the Nanolithography Toolbox manual with this page. Rounded rectangle constructor **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 $(r_x$ and $r_y)$ and are rendered using the specified **shapeReso** resolution parameter (see section 0.9.4). Rotation of the rounded rectangle is about the lower left corner (x_{LL}, y_{LL}) . Rounded rectangles defined using **roundRectC** are of length L , height H and are centered at (x_C, y_C) .

x_{LL}	y_{LL}	L	H	r_x	r_y	$\theta_{(x_{LL}, y_{LL})}$	roundRect
x_C	y_C	L	H	r_x	r_y	$\theta_{(x_C, y_C)}$	roundRectC

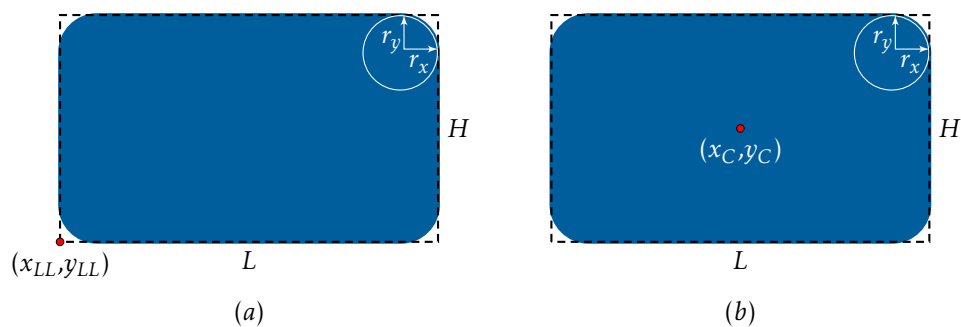


Figure 135: Rounded rectangle constructors defined by (a) lower left corner (**roundRect**) and (b) center coordinates (**roundRectC**).

0.5.5.8 Rounded Rectangle Outline

Below are two versions of the outline rounded rectangle structures defined by the lower left and center coordinates. The defining shape parameters are identical to ones described in the previous section with the addition of an outline width w_o .

x_{LL} y_{LL} L H r_x r_y w_o $\theta_{(x_{LL},y_{LL})}$ **roundRectOutline**

x_C y_C L H r_x r_y w_o $\theta_{(x_C,y_C)}$ **roundRectOutlineC**

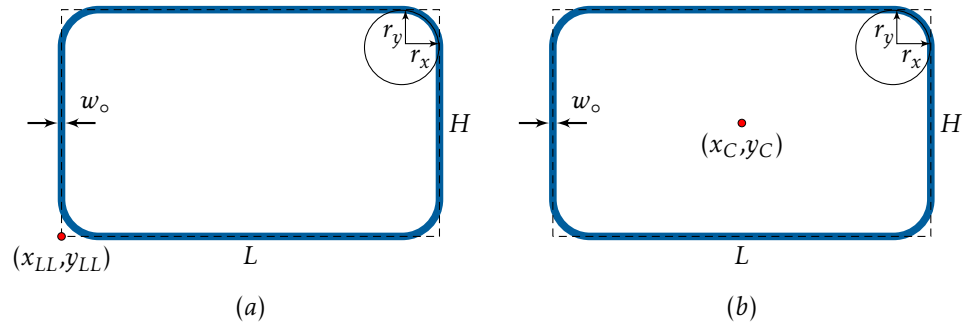


Figure 136: Rounded rectangle outline constructors defined by (a) lower left corner (**roundRectOutline**) and (b) center coordinates (**roundRectOutlineC**).

0.5.5.9 Rounded Rectangle Dashed

Below are two versions of the dashed rounded rectangle structures defined by the lower left and center coordinates. The defining shape parameters are identical to ones described in the previous section with the addition of a dash length and gap defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x_{LL} y_{LL} L H r_x r_y w_o d_L d_G Cap $Join$ $\theta_{(x_{LL},y_{LL})}$ **roundRectDashed**

x_C y_C L H r_x r_y w_o d_L d_G Cap $Join$ $\theta_{(x_C,y_C)}$ **roundRectDashedC**

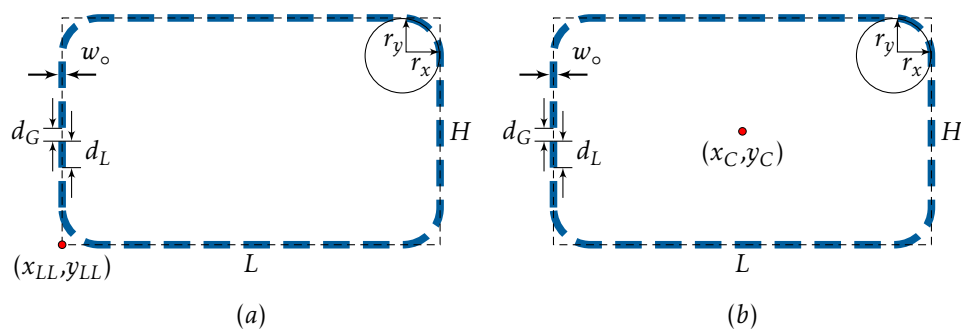


Figure 137: Rounded rectangle outline constructors defined by (a) lower left corner (**roundRectDashed**) and (b) center coordinates (**roundRectDashedC**).

0.5.5.10 Rounded Square

Below are two constructors for rounded squares defined by the lower left and center coordinates.

x_{LL} y_{LL} L r $\theta_{(x_{LL}, y_{LL})}$ **roundSquare**

x_C y_C L r $\theta_{(x_C, y_C)}$ **roundSquareC**

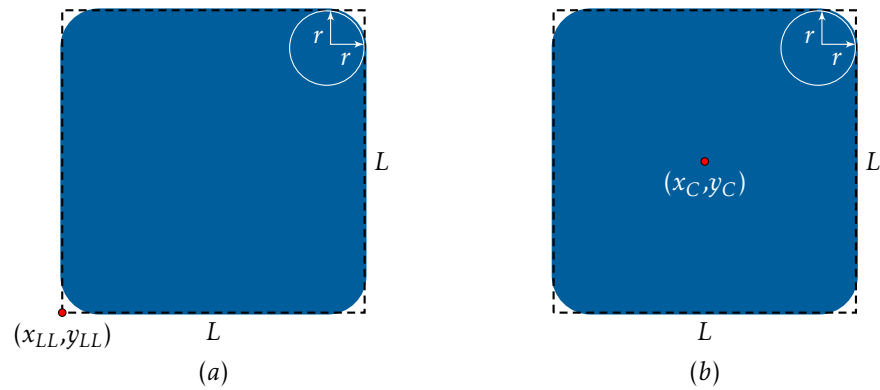


Figure 138: Rounded square constructors defined by (a) lower left corner (**roundSquare**) and (b) center coordinates (**roundSquareC**).

0.5.5.11 Rounded Square Outline

Below are two constructors for rounded squares defined by the lower left and center coordinates. The defining shape parameters are identical to ones described in the previous section with the addition of an outline width w_o .

x_{LL} y_{LL} L r w_o $\theta_{(x_{LL}, y_{LL})}$ **roundSquareOutline**

x_C y_C L r w_o $\theta_{(x_C, y_C)}$ **roundSquareOutlineC**

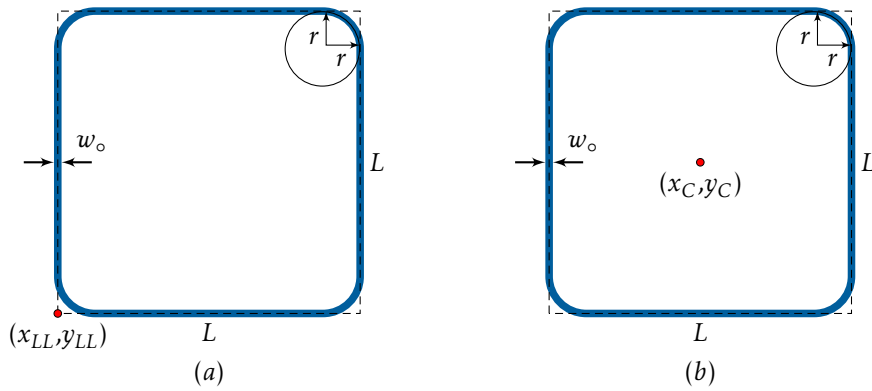


Figure 139: Rounded square constructors defined by (a) lower left corner (**roundSquareOutline**) and (b) center coordinates (**roundSquareOutlineC**).

0.5.5.12 Rounded Square Dashed

Below are two constructors for rounded squares defined by the lower left and center coordinates. The defining shape parameters are identical to ones described in the previous section with the addition of a dash length and gap defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

x_{LL} y_{LL} L r w_o d_L d_G Cap $Join$ $\theta_{(x_{LL},y_{LL})}$ **roundSquareDashed**

x_C y_C L r w_o d_L d_G Cap $Join$ $\theta_{(x_C,y_C)}$ **roundSquareDashedC**

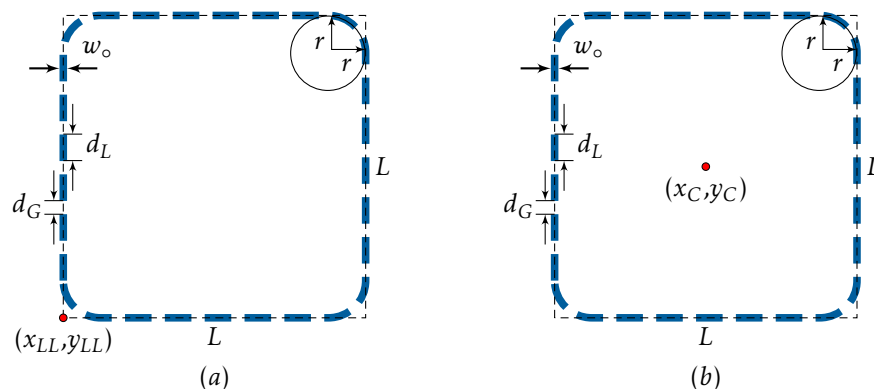


Figure 140: Rounded square constructors defined by (a) lower left corner (**roundSquareDashed**) and (b) center coordinates (**roundSquareDashedC**).

0.6 Alignment Elements - CNST Special Scripts

0.6.1 Box in box alignment marks

The `alignBB` constructor defines a box-in-box lithographic alignment mark centered around (x, y) . The outer box of width w_{a1} with a central hole of width w_{a2} is cast into a GDS layer L_a . The second level box of width w_b is cast into a GDS layer L_b .

x y w_{a1} w_{a2} w_b L_a L_b $\theta_{(x,y)}$ `alignBB`

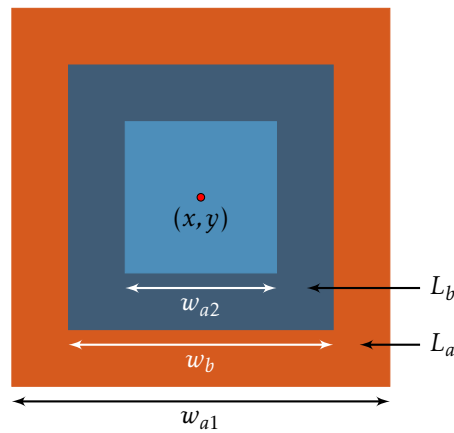


Figure 141: Box-in-box lithographic alignment mark centered around (x, y) .

0.6.2 Arrows

0.6.2.1 Arrow Shapes

Update 2016 release of the Nanolithography Toolbox manual with this page. 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.

$\langle x$	y	W_a	L_a	W		$\theta_{(x,y)}$	arrowHead>
$\langle x$	y	W_a	L_a	W	L	$\theta_{(x,y)}$	arrow>
$\langle x$	y	W_a	L_a	W	L	N	arrowArray>

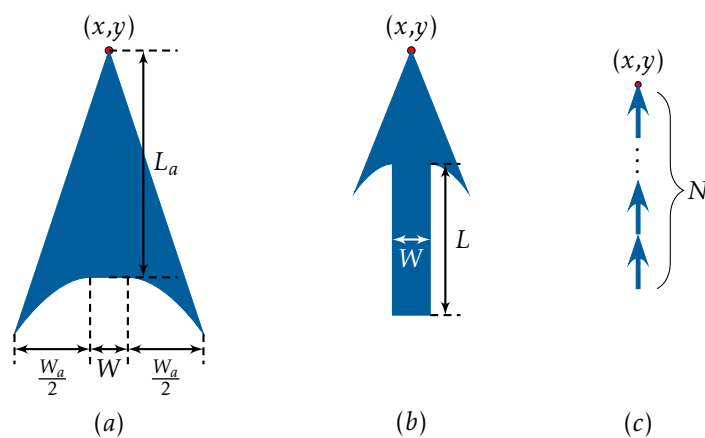


Figure 142: Example shapes illustrating various parameters from the (a) **arrowHead**, (b) **arrow** and (c) **arrowArray** constructors.

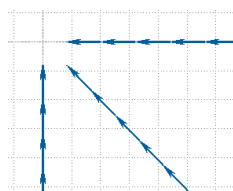


Figure 143: Rendered GDS file example with **arrowArray** structures.

0.6.2.2 Arrow Shapes - Outline and Dashed

The following constructors are outlined and dashed versions of the various arrow shapes defined in the previous section 0.6.2.1. The outline constructors are defined by the shape outline width w_o . Dash length and gap are defined by d_L and d_G , respectively. The Cap and Join parameters define the end-cap and line-join parameters.

$\langle x \ y \ W_a \ L_a \ W \ w_o \ \theta_{(x,y)} \ \text{arrowHeadOutline} \rangle$

$\langle x \ y \ W_a \ L_a \ W \ L \ w_o \ \theta_{(x,y)} \ \text{arrowOutline} \rangle$

$\langle x \ y \ W_a \ L_a \ W \ L \ N \ w_o \ \theta_{(x,y)} \ \text{arrowArrayOutline} \rangle$

$\langle x \ y \ W_a \ L_a \ W \ w_o \ d_L \ d_G \ \text{Cap} \ \text{Join} \ \theta_{(x,y)} \ \text{arrowHeadDashed} \rangle$

$\langle x \ y \ W_a \ L_a \ W \ L \ w_o \ d_L \ d_G \ \text{Cap} \ \text{Join} \ \theta_{(x,y)} \ \text{arrowDashed} \rangle$

$\langle x \ y \ W_a \ L_a \ W \ L \ N \ w_o \ d_L \ d_G \ \text{Cap} \ \text{Join} \ \theta_{(x,y)} \ \text{arrowArrayDashed} \rangle$

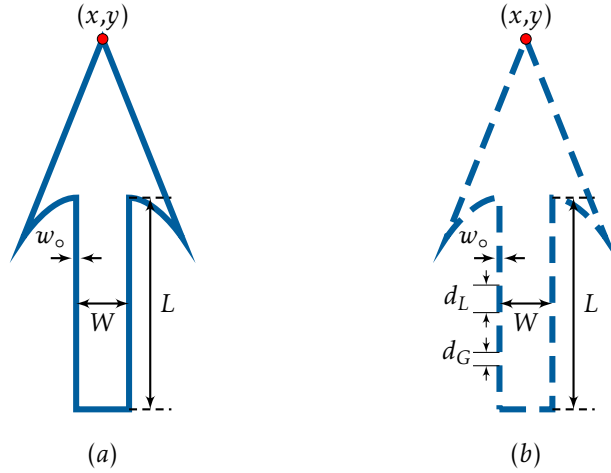


Figure 144: Example shapes illustrating various parameters from the (a) **arrowOutline** and (b) **arrowDashed** constructors.

0.6.3 Axes

0.6.3.1 Axes

The constructors defined below are useful for labeling and locating features of interest. For instance, the axes could be used to locate small area, electron beam lithographically defined dose matrices. The two versions below have arrow heads of length L_a and width W_a that extends beyond the arrow widths W , W_1 , and W_2 . The arrow heads are defined in section 0.6.2.1.

`<x y W L W_a L_a $\theta_{(x,y)}$ axes>`
`<x y W_1 L_1 W_2 L_2 W_a L_a $\theta_{(x,y)}$ axesV2>`

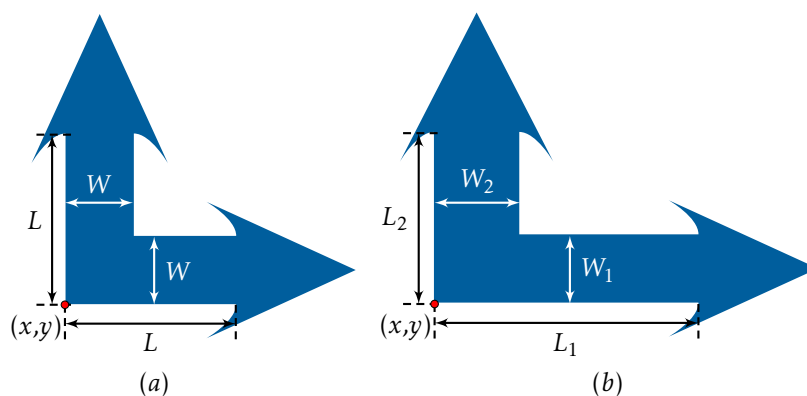


Figure 145: Axes created using the (a) `axes` and (b) `axesV2` constructors.

0.6.3.2 Axes - Outline

Below constructors for outlined axes shapes. The defining shape parameters are identical to ones described in the previous section with the addition of an outline width w_o .

$\langle x \ y \ W \ L \ W_a \ L_a \ w_o \ \theta_{(x,y)} \ \text{axesOutline} \rangle$

$\langle x \ y \ W_1 \ L_1 \ W_2 \ L_2 \ W_a \ L_a \ w_o \ \theta_{(x,y)} \ \text{axesOutlineV2} \rangle$

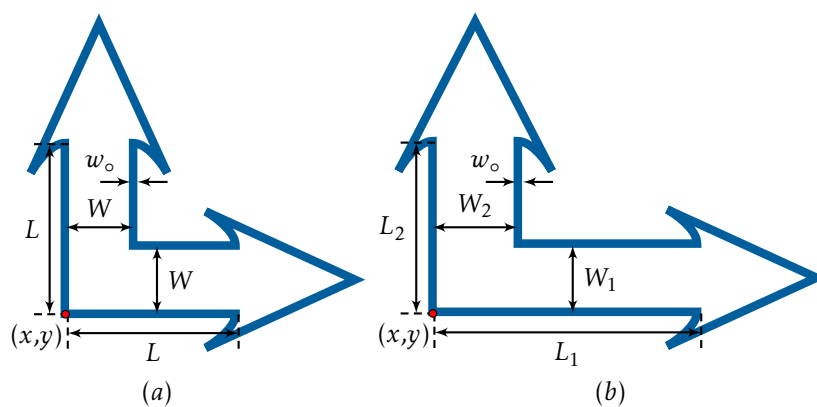


Figure 146: Axes created using the (a) `axesOutline` and (b) `axesOutlineV2` constructors.

0.6.3.3 Axes - Dashed

Below constructors for dashed axes shapes. The defining shape parameters are identical to ones described in the previous section with the addition of a dash length and gap defined by d_L and d_G , respectively. The Cap and Join, shown in Figure 116, define the end-cap and line-join parameters.

$\langle x \ y \ W \ L \ W_a \ L_a \ w_o \ d_L \ d_G \ Cap \ Join \ \theta_{(x,y)} \ \text{axesDashed} \rangle$

$\langle x \ y \ W_1 \ L_1 \ W_2 \ L_2 \ W_a \ L_a \ w_o \ d_L \ d_G \ Cap \ Join \ \theta_{(x,y)} \ \text{axesDashedV2} \rangle$

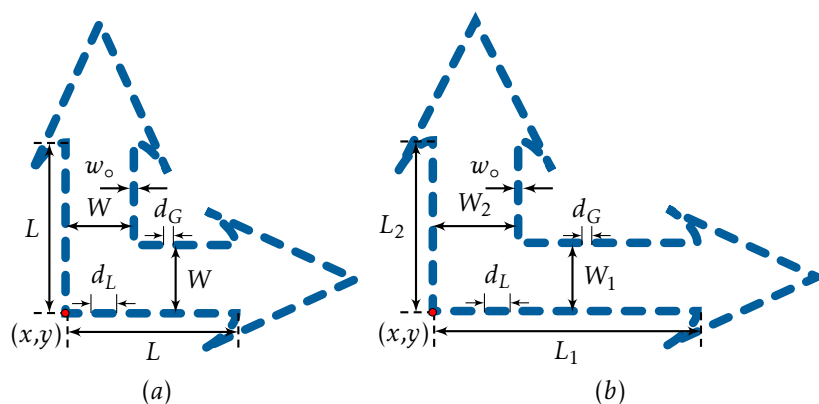


Figure 147: Axes created using the (a) *axesDashed* and (b) *axesDashedV2* constructors.

0.7 MEMS-NEMS - CNST Special Scripts

0.7.1 Beam With Curved Ends

Beam with a curved end defined by a position (x, y) , length L , width W , and circle of radius r with number of vertices to N_{sides} . The beam is defined by (x, y) on the left end or the center of the beam (constructor suffix **C**). The vectorized constructors are defined with a suffix **V**. Here number of vertices is defined by the **shapeReso** parameter.

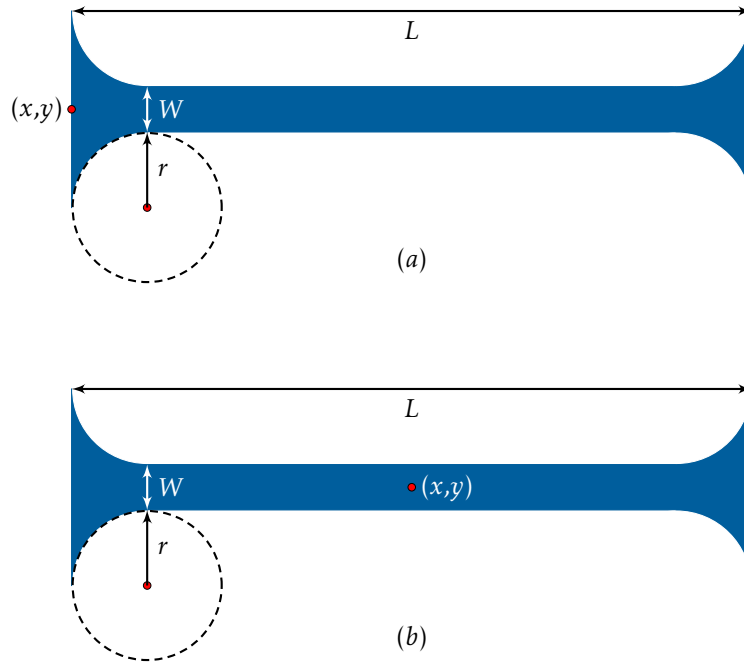


Figure 148: Beam with curved ends defined with a circle of radius r with (x, y) defined at the (a) left hand side and (b) at the center of the beam.

x	y	L	W	r	N_{sides}	$\theta_{(x,y)}$	beamCurvedEnds
x	y	L	W	r		$\theta_{(x,y)}$	beamCurvedEndsV
x	y	L	W	r	N_{sides}	$\theta_{(x,y)}$	beamCurvedEndsC
x	y	L	W	r		$\theta_{(x,y)}$	beamCurvedEndsCV

0.7.2 Bolometers

L-shaped bolometer structure. Here the vertices represented by r are defined by the number of sides parameter N_{sides} . The radii r_1 and r_2 rendering resolution is defined by the [shapeReso](#) parameter.

x y w_1 w_2 w_3 g_1 g_2 g_3 g_4 L_1 L_2 r_1 r_2 r N_{sides} a b c d e f L_a L_b L_c L_d L_e L_f $\theta_{(x,y)}$ [bolometerL2](#)

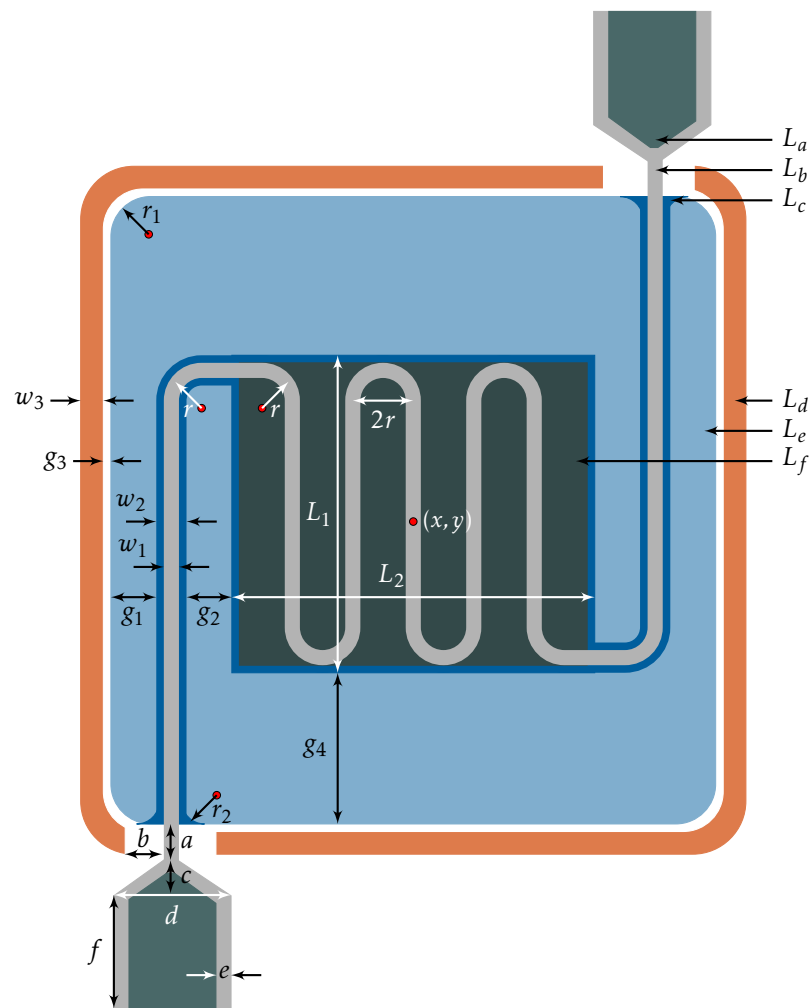


Figure 149: L-shaped bolometer structure.

0.7.3 Suspended Ring Array

Vertices of the curved suspended ring array segments are defined by the rendering resolution `shapeReso` parameter.

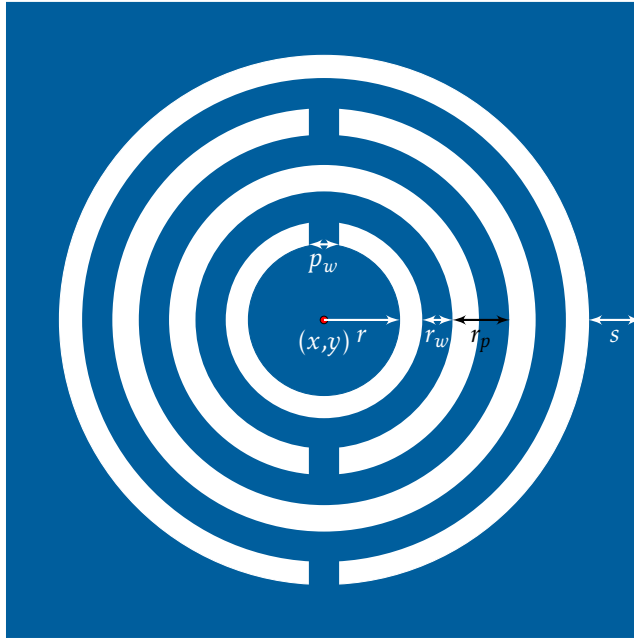


Figure 150: Suspended ring array example.

x y r r_w r_p N_{rings} p_w s $\theta_{(x,y)}$ `memsRingsV1`

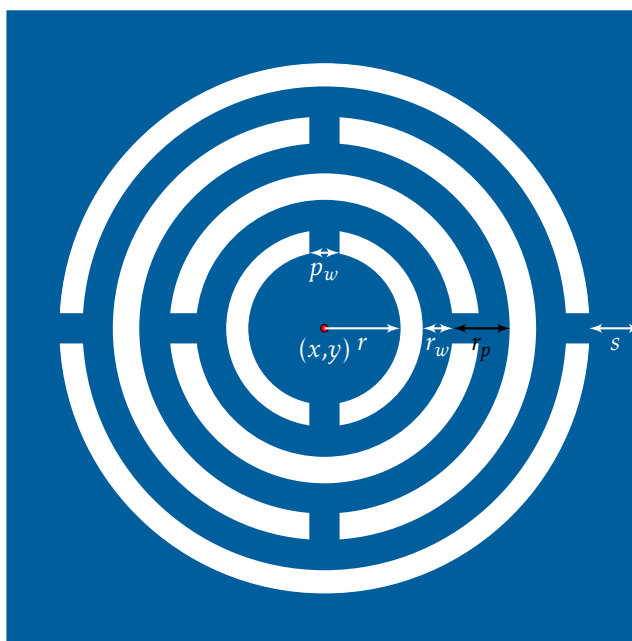


Figure 151: Suspended ring array example.

x y r r_w r_p N_{rings} p_w s $\theta_{(x,y)}$ `memsRingsV2`

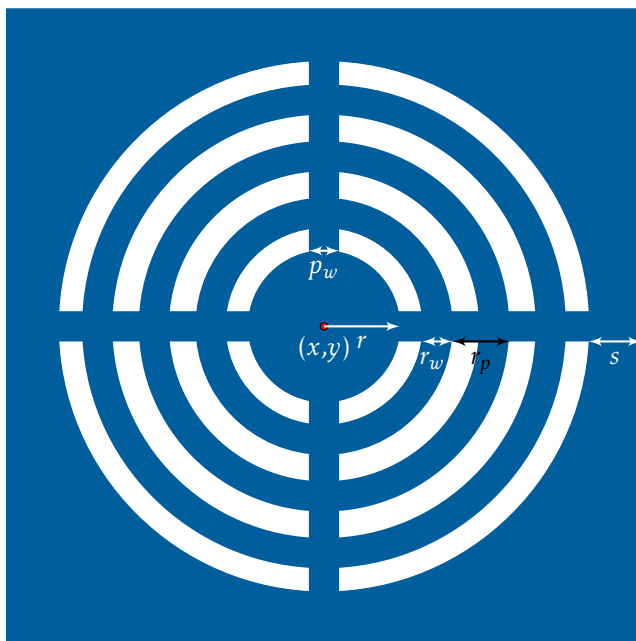


Figure 152: *Suspended ring array example.*

x y r r_w r_p N_{rings} p_w s $\theta_{(x,y)}$ **memsRingsV3**

0.8 General Area Operations

0.8.1 General Area - Outline

In a similar manner to boolean operations, shapes residing in a particular GDS layer are extracted and stored into a general area object. The string name of the object, defined by the parameter `genAreaName`, must be constructed from a continuous set of characters and must not contain any spaces or tabs. As seen in the boolean operation section, the general area extracted from a struct is accomplished by the following constructor:

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

Once the `genAreaName` is extracted, then the shape can be formed into an outline or dash-outline shape of width w_o and cast into a particular GDS layer (`resultGDSLayer`). The dashed shapes have additional parameters defining the dash length (d_L), dash gap (d_G), end-cap (`Cap`) and line-join (`Join`) parameters. The `Cap` and `Join` parameter options are illustrated in Figure 116. Since most shapes are vector defined, the `shapeReso` parameter defines the rendering resolution. The following are the outline and dash-outline constructors:

```
<genAreaName resultGDSLayer  $w_o$  OUTLINE>
```

```
<genAreaName resultGDSLayer  $w_o$   $d_L$   $d_G$  Cap Join DASHED>
```

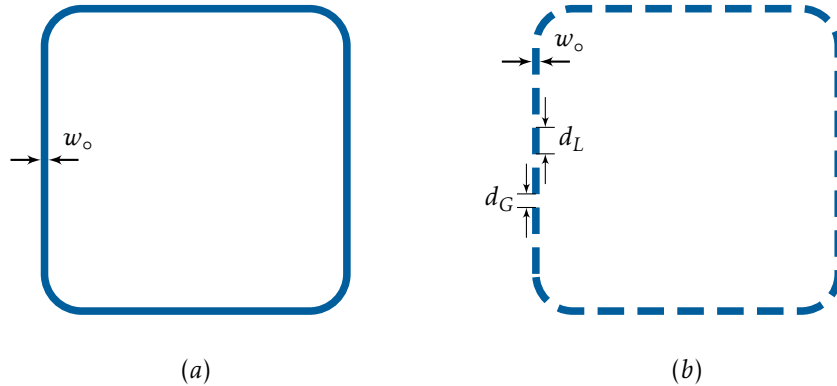


Figure 153: Example of (a) the (**OUTLINE**) and (b) (**DASHED**) constructors.

The following example, shown in figure 154, creates a wavy circular structure using the **torusWaveIn**. The structure is extracted and stored into a **genArea** object. The GDS area object is then cast to an outline and a dashed shape.

```

0.001 gdsReso
0.001 shapeReso

4 layer

#create wavy circular structure
<wavyStructure struct>
0 0 0 88 10 14 400 0 torusWaveIn
# extract area
<areaNameWavyTorus wavyStructure 4 genArea>

<outline struct>
# cast to outline GDS shape
<areaNameWavyTorus 4 2 OUTLINE>
<dashed struct>
# cast to dashed GDS shape
<areaNameWavyTorus 4 2 10 8 1 1 DASHED>

```

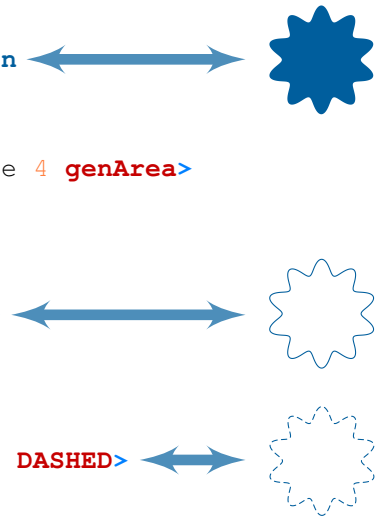


Figure 154: Scripting example of the **OUTLINE** and **DASHED** constructors. The arrows relate the script code that's used to generate the rendered shape.

0.9 Interface Functions



0.9.1 Reading-Importing GDS Files

The **readGDS** constructor imports a GDS file and allows user defined objects to be placed within imported GDS structures/cells. Repeated use of the **readGDS** constructor allows the import of multiple GDS files. The first constructor parameter is the name of the imported GDS file. The filename constructor parameter must NOT have the '.gds' extension.



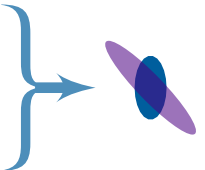
```
<gdsFileNameWithoutExtension readGDS>
```



```
# The imported readGDSTestFile.gds  
# contains a "top" structure/cell.  
<readGDSTestFile readGDS>
```



```
# This initializes the "top" structure  
# and adds new objects to the cell.  
<top struct>  
111 layer  
0 0 8 40 44 ellipseVector  
88 layer  
95 0 20 50 44 rectangleC
```



```
# GDS objects are added to "newStruct",  
# a newly created structure/cell.  
<newStruct struct>  
222 layer  
0 0 40 80 0 rectangleC
```





Figure 155: Scripting example of the **readGDS** constructor. GDS file *readGDSTestFile.gds* contains a *top* GDS struct. The *top* GDS struct is initialized and two shapes are added (*ellipseVector* and *rectangleC*) to the existing GDS struct. A *rectangleC* structure is then added to a newly created GDS struct.

0.9.2 Reading-Importing and Parsing GDS Files

The `readParseGDS` constructor first imports then parses the contents a GDS file. The contents of the imported GDS file are stored in the output ASCII log file. Following parsing, in an identical manner as the the `readParseGDS` constructor, this constructor further allows adding objects to be imported GDS structures/cells. The first constructor parameter is the name of the imported GDS file. The filename constructor parameter must NOT have the '.gds' extension.

<gdsFileNameWithoutExtension `readParseGDS`>

```
*****
GDS PARSE START : readGDSTestFile.gds
*****

HEADER  0 3      (6)
BGNLIB  0 9 0 12 0 5 0 14 0 55 0 48 0 98 0 12 0 5 0 14 0 55 0 48      (34)
LIBNAME  scriptGDS.DB      (50)
UNITS  1.0E-6 1.0E-9      (70)
BGNSTR  0 9 0 12 0 5 0 14 0 55 0 48 0 98 0 12 0 5 0 14 0 55 0 48      (98)
STRNAME  "topNOT"      (106)
BOUNDARY      (110)
LAYER  0 7      (116)
DATATYPE  0 77      (122)
XY  23 point(s)
1423 -19796 -1423 -19796 -4154 -18193 -6549 -15115 -8413 -10813 -9595 -5635
-10000 0 -9595 5635 -8413 10813 -6549 15115 -4154 18193 -1423 19796 1423
19796 4154 18193 6549 15115 8413 10813 9595 5635 10000 0 9595 -5635 8413
-10813 6549 -15115 4154 -18193 1423 -19796      (310)
ENDEL      (314)
```

Figure 156: Log file output example of a parsed GDS file using the `readParseGDS` constructor. The parsed GDS file name is displayed in the header. Directly below are ASCII representations of the parsed GDS file. This example does not include the parsed contents of the entire GDS file. The end of the parsed section contains a comment `GDS PARSE END` enclosed by a row of asterisk characters.

0.9.3 Exporting shape vertices - POINTSXY

The **POINTSXY** constructor enables the ASCII export of shapes vertices defined by GDS layers within a user defined GDS struct. Each vertex is represented by space delimited x and y coordinate values. The procedure initiates with enabling the export with the **POINTSXY** command. Then a GDS struct, containing shapes of interest, is selected using the **pointsXyStruct** command. The **pointsXyLayers** constructor is then used to define the GDS layers of shapes to be exported. Shapes within the specified GDS struct and layers are then exported into an ASCII file with an extension `.ptsXY`. Coordinate values are in units of micrometers. The export resolution (i.e. number of digits following the decimal point) is defined by the **gdsReso** parameters. The following two export options are available:

- 1) By default, shape boundaries are exported as a set of x y values with a space delimiter. The end of each shape is marked with a line-feed end-of-line termination (Figure 158).
- 2) The **pointsXyColumn** constructor formats the export such that each line contains a vertex represented by a set of x and y coordinates. The end of each shape is marked by an extra blank line (Figure 159). This option is useful when plotting shapes in matplotlib, Excel, Matlab and other plotting packages.

Constructors for exporting shape vertices:

POINTSXY

pointsXyColumn

<GDS StructName pointsXyStruct>

$L_1 L_2 L_3 L_4 \dots L_n$ pointsXyLayers

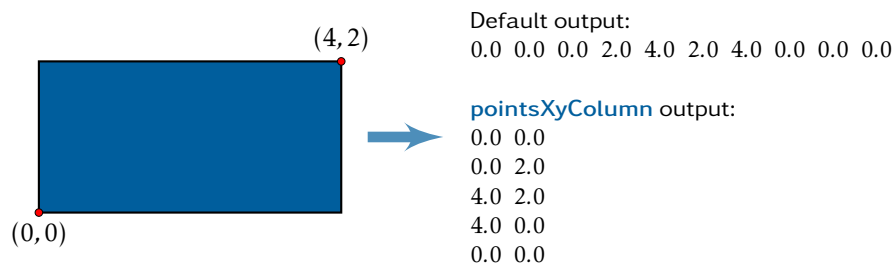


Figure 157: **POINTSXY** export example of a rectangle.

```

0.001 gdsReso
0.001 shapeReso

POINTSXY
<testShapes pointsXyStruct>
1 2 4 pointsXyLayers

<testShapes struct>
1 layer
0 0 10 4 0 rectangle
2 layer
11 0 4 4.4 0 rectangleLH
4 layer
0 0 0.5 2 7 0 ellipse
5 layer
2 0 1 1 22 0 ellipse

<moreTestShapes struct>
6 layer
0 4 1 1 10 0 ellipse

<evenMoreTestShapes struct>
7 layer
2 2.2 1 0.5 8 0 ellipse

```

(a)

```

0.0 0.0 0.0 4.0 10.0 4.0 10.0 0.0 0.0 0.0
11.0 0.0 11.0 4.4 15.0 4.4 15.0 0.0 11.0 0.0
-0.111 -1.95 -0.45 -0.868 -0.45 0.868 -0.111 1.95 0.312 1.564 0.5 0.0 0.312 -1.564 -0.111 -1.95

```

(b)

Figure 158: *POINTSXY* example (a) script and (b) default ASCII output.

```

0.001 gdsReso
0.001 shapeReso

POINTSXY
pointsXyColumn
<testShapes pointsXyStruct>
1 2 pointsXyLayers

<testShapes struct>
1 layer
0 0 10 4 0 rectangle
2 layer
11 0 4 4.4 0 rectangleLH
4 layer
0 0 0.5 2 7 0 ellipse

<moreTestShapes struct>
6 layer
0 4 1 1 10 0 ellipse

```

(a)

```

0.0 0.0
0.0 4.0
10.0 4.0
10.0 0.0
0.0 0.0

```

```

11.0 0.0
11.0 4.4
15.0 4.4
15.0 0.0
11.0 0.0

```

(b)

Figure 159: *POINTSXY* example (a) script and (b) *pointsXyColumn* ASCII output.

0.9.4 Shape Resolution

Update 2016 release of the Nanolithography Toolbox manual with this page. The method sets the rendering resolution for subsequent shapes (primarily vector defined shapes, S-Bends, Y-Bends, postscript file conversions, Text elements, etc). Default global value is $0.001\mu\text{m}$. 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.

shapeResolution **shapeReso**

0.001 **shapeReso** defines a $0.001\mu\text{m}$ rendering resolution.

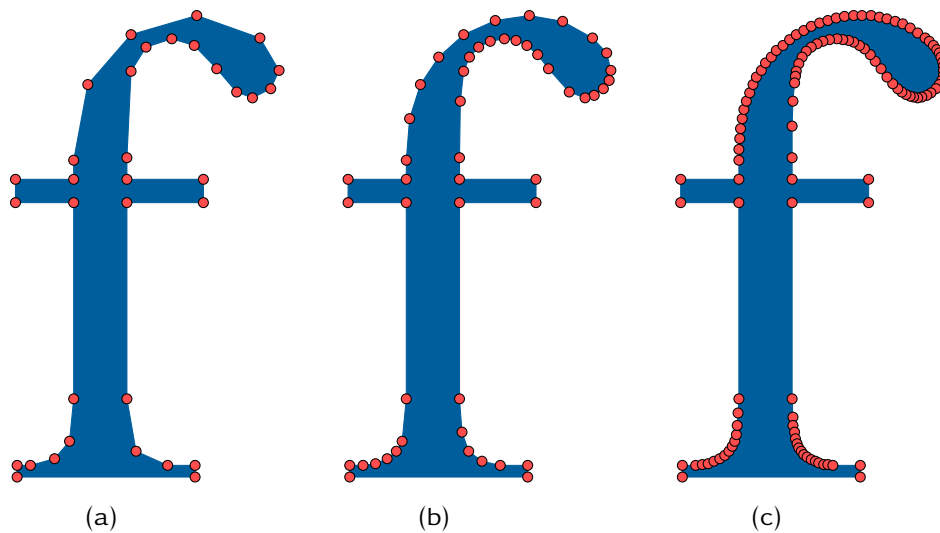


Figure 160: Example illustrating rendering of a vectorized text object at *shapeReso* values of (a) $1\mu\text{m}$ (b) $0.1\mu\text{m}$ and (c) $0.01\mu\text{m}$.

0.9.5 Font Outline Width

The method sets the font outline width (in micrometers). This is used for outline text and label objects (see sections 27 and 0.3.7). Default global value is $0.1\mu\text{m}$. This value could be changed for each outlined object

fontOutlineWidth **fontOutline**

0.400 **fontOutline** defines a $0.400\mu\text{m}$ font outline width.

0.10 Path Length Calculations

0.10.1 Path Length Calculations of NanoPhotonic Elements

Variety of nanophotonic elements could be stored into a path length container. The path length calculator is activated with the `pathLengthStart` constructor with the `pathLengthName` as the distinct parameter that defines the path length container. The `pathLengthName` parameter is a continuous string that defines the stored path length segments. Various nanophotonic segments (see list of allowed nanophotonic elements below) are then placed into the container. The container is then closed with the `pathLengthEnd` statement. Length of each segment is calculated and the total length, defined by the `pathLengthName`, is stored into the output log file.

```
<pathLengthName    pathLengthStart>
```

Place nanophotonic elements here

```
<pathLengthStart>
```

The following is a list of allow nanophotonic elements for path length calculation:

`bezierCurve`, `bezierCurveInv`, `bezierCurveInvSlot`, `waveguide`, `waveguideSlot`, `waveguideInv`, `waveguideInvSlot`, `wgExpander`, `linearTaper`, `linearTaperSlot`, `linearTaperInvSlot`, `exponentialTaper`, `exponentialTaperInv`, `exponentialTaperInvSlot`, `sBendTaper`, `sBendFunnel`, `sBend`, `sBendLH`, `sBendInv`, `sBendInvSlot`, `yBend`, `yBendLH`, `yBendInv`, `yBendInvSlot`, `yBend90`, `yBendInv90`, `90degreeBend`, `90degreeBendLH`, `90degreeBendInv`, `90degreeBendInvSlot`, `180degreeBend`, `180degreeBendInv`, `180degreeBendInvSlot`, `spiralDelayLineArch`, `spiralDelayLineArchInv`, `spiralDelayLineFermat`, `spiralDelayLineFermatInv`

Figure 161 shows three path length segments. The first path length container named `CubicBezierPaperClipVeryLongPathName` includes a variety of connected nanophotonic components. The second and third examples have single waveguide and s-bend elements. The output log file (Figure 162) shows the calculated path lengths of the three containers, i.e. `CubicBezierPaperClipVeryLongPathName`, `singleWaveGuideInSameCell` and `SingleSBendLHPath`. The last two structures are not enclosed within the path length constructors and consequently the path lengths are not calculated in the resulting log file.

A more comprehensive example file `scriptingPathLength.cnst` is included in the release package. The example contains a collection of path length calculations for a variety of nanophotonic structures. The output log file `scriptingPathLength.log` includes the path length results of each segment.


```

<CubicBezierPaperClipVeryLongPathName pathLengthStart>
<bezierCurvePaperClips struct>
<500 -400 0 -400 1 0 0 0 waveguide>
<0 -400 -50 -350 1 0 90degreeBend>
<-50 -350 0 -350 1 90 0 0 waveguide>
<0 -250 -50 -300 1 0 90degreeBend>
<0 -250 300 -350 1 0 sBend>
<500 -300 300 -350 1 0 0 0 waveguide>
<pathLengthEnd>

<singleWaveGuideInSameCell pathLengthStart>
<500 -100 0 -100 1 0 0 0 waveguide>
<pathLengthEnd>

# test of various bezier curves
<sBendTest struct>
<SingleSBendLHPath pathLengthStart>
<0 0 10 10 0.5 0 sBendLH>
<pathLengthEnd>

# the following devices are not included in the
# path length calculations
<bezierAndSBend struct>
<0 200 -100 200 -100 350 0 350 1 0 bezierCurve>
<0 350 300 250 1 0 sBend>

<singleWaveguide struct>
<500 500 0 500 1 0 0 0 waveguide>

```

Figure 161: Multiline *cnst* script file example using the path length *pathLengthStart* and *pathLengthEnd* constructors.

```

Segment "CubicBezierPaperClipVeryLongPathName" path length = 1233.2266 micrometers
Segment "singleWaveGuideInSameCell" path length = 500 micrometers
Segment "SingleSBendLHPath" path length = 14.9525 micrometers
COMPLETED!

```

Figure 162: Logfile output showing three path length calculations from Figure 161 *cnst* script file example.

0.11 CNST Special Reference Methods

Below are reference method headers for special methods released within CNST.

0.11.1 CNST Special - Miscellaneous Object Methods

```
// *****  
//  
// Nanophotonics GDS Area and Struct Methods  
//  
// *****  
  
// 180 degree bend with taper  
public static GArea create180degreeBendT(double x, double y, double L1,  
    double L2, double diameter, double W,  
    double Wtaper, int numSides, double THETA, int gdsLayer)  
  
// waveguide connected to a linear taper  
public static GArea createWgLinearTaper(double x, double y, double L,  
    double W, double Ltaper, double Wtaper, double THETA, int gdsLayer)  
  
// directionalCoupler1i  
public static GArea createDirectionalCoupler1i(double x, double y,  
    double L1, double w1, double L2, double L3, double w2, double g,  
    double r, int numSides, double THETA, int gdsLayer)  
  
// directionalCoupler2i  
public static GArea createDirectionalCoupler2i(double x, double y,  
    double L1, double w1, double L2, double L3, double w2, double g,  
    double r, int numSides, double THETA, int gdsLayer)  
  
// directionalCoupler3i  
public static GArea createDirectionalCoupler3i(double x, double y,  
    double w, double g, double L1, double L2, double r, int numSides,  
    double THETA, int gdsLayer)  
  
// directionalCoupler4i  
public static GArea createDirectionalCoupler4i(double x, double y,  
    double w, double g, double L1, double LB, double HB, double THETA,  
    int gdsLayer)  
  
// raceTrackWG  
public static GArea createRaceTrackWG(double x, double y, double rL,  
    double rW, double rR, int NR, double L1, double g1, double w1,  
    double L2, double g2, double w2, double THETA, int ENDCAP, int  
    gdsLayer)  
  
// raceTrackWGi  
public static GArea createRaceTrackWGi(double x, double y, double rL,  
    double rW, double rR, int NR, double L1, double g1, double w1,  
    double we1, double L2, double g2, double w2, double we2, double  
    THETA, int ENDCAP, int gdsLayer)
```

```

// raceTrackWGpi
public static GArea createRaceTrackWGpi(double x, double y, double rL,
    double rW, double re, double rR, int NR, double L1, double g1,
    double w1, double we1, double L2, double g2, double w2, double we2,
    double THETA, int ENDCAP, int gdsLayer)

// raceTrackWGs
public static GArea createRaceTrackWGs(double x, double y, double rL,
    double rW, double rR, int NR, double L1, double g1, double w1,
    double r1, int N1, double L2, double g2, double w2, double r2, int
    N2, double THETA, int ENDCAP, int gdsLayer)

// raceTrackWGsi
public static GArea createRaceTrackWGsi(double x, double y, double rL,
    double rW, double rR, int NR, double L1, double g1, double w1,
    double we1, double r1, int N1, double L2, double g2, double w2,
    double we2, double r2, int N2, double THETA, int ENDCAP, int
    gdsLayer)

// raceTrackWGspi
public static GArea createRaceTrackWGspi(double x, double y, double rL,
    double rW, double re, double rR, int NR, double L1, double g1,
    double w1, double we1, double r1, int N1, double L2, double g2,
    double w2, double we2, double r2, int N2, double THETA, int ENDCAP,
    int gdsLayer)

// taperSB – parameters: x y bW bH rb bLyr wT1 wT2 hT rT H1 w1 r1 ....
// Hi wi ri THETA
// box layer bLyr set as a double -> will be converted to an int
public static ArrayList<GArea> createTaperSB(ArrayList<Double> al, int
    gdsLayer)

// taperSB – parameters: x y bW bH rb bLyr wT1 wT2 hT rT H1 w1 r1 ....
// Hi wi ri THETA
// box layer bLyr set as a double -> will be converted to an int
public static void createTaperSB(Struct currentStruct, ArrayList<Double>
    al, int gdsLayer)

// taperSBinv – parameters: x y bW bH rb wT1 wT2 hT rT H1 w1 r1 .... Hi
// wi ri THETA
public static ArrayList<GArea> createTaperSBinv(ArrayList<Double> al,
    int gdsLayer)

// taperSBinv – parameters: x y bW bH rb wT1 wT2 hT rT H1 w1 r1 .... Hi
// wi ri THETA
public static void createTaperSBinv(Struct currentStruct, ArrayList<
    Double> al, int gdsLayer)

// taperSBinv – parameters: x y bW bH rb wT1 wT2 hT rT H1 w1 r1 .... Hi
// wi ri THETA
public static GArea createTaperSupport(ArrayList<Double> al, int
    gdsLayer)

// 1X2MMI – 1X2 Multimode Interface Coupler
public static GArea create1X2MMI(double x, double y, double wi, double
    Li, double Hi, double w, double L, double wo1, double Lo1, double
    Ho1, double wo2, double Lo2, double Ho2, double THETA, int gdsLayer)

```

```

    )

// 1X2MMI – 1X2 Multimode Interface Coupler
public static void create1X2MMI(Struct currentStruct, double x, double
    y, double wi, double Li, double Hi, double w, double L, double wo1,
    double Lo1, double Ho1, double wo2, double Lo2, double Ho2, double
    THETA, int gdsLayer)

// 1X2MMItaper – 1X2 Multimode Interface Coupler with taper
public static GArea create1X2MMItaper(double x, double y, double wi,
    double Li, double wti, double Lti, double Hi, double w, double L,
    double wo1, double Lo1, double wto1, double Lto1, double Ho1,
    double wo2, double Lo2, double wto2, double Lto2, double Ho2,
    double THETA, int gdsLayer)

// 1X2MMItaper – 1X2 Multimode Interface Coupler with taper
public static void create1X2MMItaper(Struct currentStruct, double x,
    double y, double wi, double Li, double wti, double Lti, double Hi,
    double w, double L, double wo1, double Lo1, double wto1, double
    Lto1, double Ho1, double wo2, double Lo2, double wto2, double Lto2,
    double Ho2, double THETA, int gdsLayer)

// mXnMMItaper – nXm Multimode Interface Coupler with taper
public static GArea createMXnMMItaper(ArrayList<Double> alD, int
    gdsLayer)

// mXnMMItaper – nXm Multimode Interface Coupler with taper
public static void createMXnMMItaper(Struct currentStruct, ArrayList<
    Double> alD, int gdsLayer)

// createWaveguideWave
public static GArea createWaveguideWave(double x, double y, double L1,
    double L2, double width, double length, double amplitude, double
    THETA, int gdsLayer, double shapeReso)

// createWaveguideWaveArray
public static void createWaveguideWaveArray(Struct currentStruct,
    String waveguideStructName, double x, double y, double L1, double
    L2, double width, double length, double amplitude, int numElements,
    int gdsLayer, double shapeReso)

//
// Vectorized Disc–Ring Architectures
//

// discInfiniteV
public static void createDiscInfiniteV(Struct currentStruct, double x,
    double y, double radius, double Wr, double g, double L, double W,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discInfiniteV
public static GArea createDiscInfiniteV(double x, double y, double
    radius, double Wr, double g, double L, double W, boolean ENDCAP,
    int gdsLayer, int dataType, double shapeReso)

// ringInfiniteV

```

```

public static void createRingInfinite(Struct currentStuct, double x,
    double y, double radius, double Wr, double g, double L, double W,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringInfiniteV
public static GArea createRingInfinite(double x, double y, double
    radius, double Wr, double g, double L, double W, boolean ENDCAP,
    int gdsLayer, int dataType, double shapeReso)

// discInfDSV
public static void createDiscInfDSV(Struct currentStuct, double x,
    double y, double radius, double Wr, double g1, double L1, double W1
    , double g2, double L2, double W2, boolean ENDCAP, int gdsLayer,
    int dataType, double shapeReso)

// discInfDSV
public static GArea createDiscInfDSV(double x, double y, double radius,
    double Wr, double g1, double L1, double W1, double g2, double L2,
    double W2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringInfDSV
public static void createRingInfDSV(Struct currentStuct, double x,
    double y, double radius, double Wr, double g1, double L1, double W1
    , double g2, double L2, double W2, boolean ENDCAP, int gdsLayer,
    int dataType, double shapeReso)

// ringInfDSV
public static GArea createRingInfDSV(double x, double y, double radius,
    double Wr, double g1, double L1, double W1, double g2, double L2,
    double W2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discInfiniteInvV
public static void createDiscInfiniteInvV(Struct currentStuct, double x
    , double y, double radius, double Wr, double g, double L, double W,
    double We, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discInfiniteInvV
public static GArea createDiscInfiniteInvV(double x, double y, double
    radius, double Wr, double g, double L, double W, double We, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringInfiniteInvV
public static void createRingInfiniteInvV(Struct currentStuct, double x
    , double y, double radius, double Wr, double g, double L, double W,
    double We, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringInfiniteInvV
public static GArea createRingInfiniteInvV(double x, double y, double
    radius, double Wr, double g, double L, double W, double We, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discInfInvDSV

```

```

public static void createDiscInflInvDSV(Struct currentStuct, double x,
    double y, double radius, double Wr, double g1, double L1, double W1
    , double We1, double g2, double L2, double W2, double We2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discInflInvDSV
public static GArea createDiscInflInvDSV(double x, double y, double
    radius, double Wr, double g1, double L1, double W1, double We1,
    double g2, double L2, double W2, double We2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// ringInflInvDSV
public static void createRingInflInvDSV(Struct currentStuct, double x,
    double y, double radius, double Wr, double g1, double L1, double W1
    , double We1, double g2, double L2, double W2, double We2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringInflInvDSV
public static GArea createRingInflInvDSV(double x, double y, double
    radius, double Wr, double g1, double L1, double W1, double We1,
    double g2, double L2, double W2, double We2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// discInfiniteInvPosV
public static void createDiscInfiniteInvPosV(Struct currentStuct,
    double x, double y, double radius, double Wr, double re, double g,
    double L, double W, double We, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discInfiniteInvPosV
public static GArea createDiscInfiniteInvPosV(double x, double y,
    double radius, double Wr, double re, double g, double L, double W,
    double We, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringInfiniteInvPosV
public static void createRingInfiniteInvPosV(Struct currentStuct,
    double x, double y, double radius, double Wr, double re, double g,
    double L, double W, double We, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringInfiniteInvPosV
public static GArea createRingInfiniteInvPosV(double x, double y,
    double radius, double Wr, double re, double g, double L, double W,
    double We, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discInflInvPosDSV
public static void createDiscInflInvPosDS(Struct currentStuct, double x,
    double y, double radius, double Wr, double re, double g, double L,
    double W, double We, double g2, double L2, double W2, double We2,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discInflInvPosDSV
public static GArea createDiscInflInvPosDS(double x, double y, double
    radius, double Wr, double re, double g, double L, double W, double
    We, double g2, double L2, double W2, double We2, boolean ENDCAP,

```

```

        int gdsLayer, int dataType, double shapeReso)

// ringInflnvPosDSV
public static void createRingInflnvPosDS(Struct currentStruct, double x,
    double y, double radius, double Wr, double re, double g, double L,
    double W, double We, double g2, double L2, double W2, double We2,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringInflnvPosDSV
public static GArea createRingInflnvPosDS(double x, double y, double
    radius, double Wr, double re, double g, double L, double W, double
    We, double g2, double L2, double W2, double We2, boolean ENDCAP,
    int gdsLayer, int dataType, double shapeReso)

// discSymmetricAV
public static void createDiscSymmetricAV(Struct currentStruct, double x
    , double y, double radius, double Wr, double gap, double angle,
    double W, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
    double shapeReso)

// discSymmetricAV
public static GArea createDiscSymmetricAV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double Ls,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymmetricAV
public static void createRingSymmetricAV(Struct currentStruct, double x
    , double y, double radius, double Wr, double gap, double angle,
    double W, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
    double shapeReso)

// ringSymmetricAV
public static GArea createRingSymmetricAV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double Ls,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymADSV
public static void createDiscSymADS(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double Ls, double gap2, double W2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymADSV
public static GArea createDiscSymADS(double x, double y, double radius,
    double Wr, double gap, double angle, double W, double Ls, double
    gap2, double W2, double L2, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringSymADSV
public static void createRingSymADS(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double Ls, double gap2, double W2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymADSV
public static GArea createRingSymADS(double x, double y, double radius,
    double Wr, double gap, double angle, double W, double Ls, double

```

```

gap2, double W2, double L2, boolean ENDCAP, int gdsLayer, int
dataType, double shapeReso)

// discSymAPuLV
public static void createDiscSymAPuLV(Struct currentStruct, double x,
double y, double radius, double Wr, double gap, double angle,
double W, double Ls, double gap2, double angle2, double W2, double
Ls2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymAPuLV
public static GArea createDiscSymAPuLV(double x, double y, double
radius, double Wr, double gap, double angle, double W, double Ls,
double gap2, double angle2, double W2, double Ls2, boolean ENDCAP,
int gdsLayer, int dataType, double shapeReso)

// ringSymAPuLV
public static void createRingSymAPuLV(Struct currentStruct, double x,
double y, double radius, double Wr, double gap, double angle,
double W, double Ls, double gap2, double angle2, double W2, double
Ls2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymAPuLV
public static GArea createRingSymAPuLV(double x, double y, double
radius, double Wr, double gap, double angle, double W, double Ls,
double gap2, double angle2, double W2, double Ls2, boolean ENDCAP,
int gdsLayer, int dataType, double shapeReso)

// discSymmetricLCAV
public static void createDiscSymmetricLCAV(Struct currentStruct, double
x, double y, double radius, double Wr, double gap, double Lc,
double W, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// discSymmetricLCAV
public static GArea createDiscSymmetricLCAV(double x, double y, double
radius, double Wr, double gap, double Lc, double W, double Ls,
boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymmetricLCAV
public static void createRingSymmetricLCAV(Struct currentStruct, double
x, double y, double radius, double Wr, double gap, double Lc,
double W, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// ringSymmetricLCAV
public static GArea createRingSymmetricLCAV(double x, double y, double
radius, double Wr, double gap, double Lc, double W, double Ls,
boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymLCADSV
public static void createDiscSymLCADSV(Struct currentStruct, double x,
double y, double radius, double Wr, double gap, double Lc, double W,
double Ls, double gap2, double W2, double L2, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// discSymLCADSV

```



```

public static GArea createDiscSymLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double W2, double L2, boolean ENDCAP, int gdsLayer,
    int dataType, double shapeReso)

// ringSymLCADSV
public static void createRingSymLCADSV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double W
    , double Ls, double gap2, double W2, double L2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// ringSymLCADSV
public static GArea createRingSymLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double W2, double L2, boolean ENDCAP, int gdsLayer,
    int dataType, double shapeReso)

// discSymLCAPuIV
public static void createDiscSymLCAPuIV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double
    W, double Ls, double gap2, double Lc2, double W2, double Ls2,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymLCAPuIV
public static GArea createDiscSymLCAPuIV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double Lc2, double W2, double Ls2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// ringSymLCAPuIV
public static void createRingSymLCAPuIV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double
    W, double Ls, double gap2, double Lc2, double W2, double Ls2,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymLCAPuIV
public static GArea createRingSymLCAPuIV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double Lc2, double W2, double Ls2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// discSymmetricInvAV
public static void createDiscSymmetricInvAV(Struct currentStruct,
    double x, double y, double radius, double Wr, double gap, double
    angle, double W, double We, double Ls, boolean ENDCAP, int gdsLayer
    , int dataType, double shapeReso)

// discSymmetricInvAV
public static GArea createDiscSymmetricInvAV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringSymmetricInvAV
public static void createRingSymmetricInvAV(Struct currentStruct,
    double x, double y, double radius, double Wr, double gap, double
    angle, double W, double We, double Ls, boolean ENDCAP, int gdsLayer

```

```

        , int dataType, double shapeReso)

// ringSymmetricInvAV
public static GArea createRingSymmetricInvAV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discSymInvADSV
public static void createDiscSymInvADSV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discSymInvADSV
public static GArea createDiscSymInvADSV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymInvADSV
public static void createRingSymInvADSV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringSymInvADSV
public static GArea createRingSymInvADSV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymInvAPuLV
public static void createDiscSymInvAPuLV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double angle2, double
    W2, double We2, double L2, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discSymInvAPuLV
public static GArea createDiscSymInvAPuLV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double angle2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringSymInvAPuLV
public static void createRingSymInvAPuLV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double angle2, double
    W2, double We2, double L2, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringSymInvAPuLV

```

```

public static GArea createRingSymInvAPul(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double angle2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discSymmetricInvLCAV
public static void createDiscSymmetricInvLCAV(Struct currentStruct,
    double x, double y, double radius, double Wr, double gap, double Lc
    , double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discSymmetricInvLCAV
public static GArea createDiscSymmetricInvLCAV(double x, double y,
    double radius, double Wr, double gap, double Lc, double W, double
    We, double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringSymmetricInvLCAV
public static void createRingSymmetricInvLCAV(Struct currentStruct,
    double x, double y, double radius, double Wr, double gap, double Lc
    , double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringSymmetricInvLCAV
public static GArea createRingSymmetricInvLCAV(double x, double y,
    double radius, double Wr, double gap, double Lc, double W, double
    We, double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discSymInvLCADSV
public static void createDiscSymInvLCADSV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discSymInvLCADSV
public static GArea createDiscSymInvLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymInvLCADSV
public static void createRingSymInvLCADSV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringSymInvLCADSV
public static GArea createRingSymInvLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymInvLCAPuIV

```

```

public static void createDiscSymInvLCAPuV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, double gap2, double Lc2, double W2,
    double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
    double shapeReso)

// discSymInvLCAPuV
public static GArea createDiscSymInvLCAPuV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, double gap2, double Lc2, double W2, double We2, double
    L2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymInvLCAPuV
public static void createRingSymInvLCAPuV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, double gap2, double Lc2, double W2,
    double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
    double shapeReso)

// ringSymInvLCAPuV
public static GArea createRingSymInvLCAPuV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, double gap2, double Lc2, double W2, double We2, double
    L2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymmetricInvPosAV
public static void createDiscSymmetricInvPosAV(Struct currentStruct,
    double x, double y, double radius, double Wr, double re, double gap
    , double angle, double W, double We, double Ls, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// discSymmetricInvPosAV
public static GArea createDiscSymmetricInvPosAV(double x, double y,
    double radius, double Wr, double re, double gap, double angle,
    double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringSymmetricInvPosAV
public static void createRingSymmetricInvPosAV(Struct currentStruct,
    double x, double y, double radius, double Wr, double re, double gap
    , double angle, double W, double We, double Ls, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// ringSymmetricInvPosAV
public static GArea createRingSymmetricInvPosAV(double x, double y,
    double radius, double Wr, double re, double gap, double angle,
    double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discSymInvPosADSV
public static void createDiscSymInvPosADSV(Struct currentStruct, double
    x, double y, double radius, double Wr, double re, double gap,
    double angle, double W, double We, double Ls, double gap2, double
    W2, double We2, double L2, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discSymInvPosADSV

```

```

public static GArea createDiscSymInvPosADSV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double W2, double We2, double L2
, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymInvPosADSV
public static void createRingSymInvPosADSV(Struct currentStruct, double
x, double y, double radius, double Wr, double re, double gap,
double angle, double W, double We, double Ls, double gap2, double
W2, double We2, double L2, boolean ENDCAP, int gdsLayer, int
dataType, double shapeReso)

// ringSymInvPosADSV
public static GArea createRingSymInvPosADSV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double W2, double We2, double L2
, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymInvPosAPuLV
public static void createDiscSymInvPosAPuLV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double angle, double W, double We, double Ls, double gap2, double
angle2, double W2, double We2, double L2, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// discSymInvPosAPuLV
public static GArea createDiscSymInvPosAPuLV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double angle2, double W2, double
We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// ringSymInvPosAPuLV
public static void createRingSymInvPosAPuLV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double angle, double W, double We, double Ls, double gap2, double
angle2, double W2, double We2, double L2, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// ringSymInvPosAPuLV
public static GArea createRingSymInvPosAPuLV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double angle2, double W2, double
We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// discSymmetricInvPosLCAV
public static void createDiscSymmetricInvPosLCAV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double Lc, double W, double We, double Ls, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// discSymmetricInvPosLCAV
public static GArea createDiscSymmetricInvPosLCAV(double x, double y,
double radius, double Wr, double re, double gap, double Lc, double
W, double We, double Ls, boolean ENDCAP, int gdsLayer, int dataType

```

```

, double shapeReso)

// ringSymmetricInvPosLCAV
public static void createRingSymmetricInvPosLCAV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double Lc, double W, double We, double Ls, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// ringSymmetricInvPosLCAV
public static GArea createRingSymmetricInvPosLCAV(double x, double y,
double radius, double Wr, double re, double gap, double Lc, double
W, double We, double Ls, boolean ENDCAP, int gdsLayer, int dataType
, double shapeReso)

// discSymInvPosLCADSV
public static void createDiscSymInvPosLCADSV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double Lc, double W, double We, double Ls, double gap2, double W2
, double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType
, double shapeReso)

// discSymInvPosLCADSV
public static GArea createDiscSymInvPosLCADSV(double x, double y,
double radius, double Wr, double re, double gap, double Lc, double
W, double We, double Ls, double gap2, double W2, double We2, double
L2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringSymInvPosLCADSV
public static void createRingSymInvPosLCADSV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double Lc, double W, double We, double Ls, double gap2, double W2
, double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType
, double shapeReso)

// ringSymInvPosLCADSV
public static GArea createRingSymInvPosLCADSV(double x, double y,
double radius, double Wr, double re, double gap, double Lc, double
W, double We, double Ls, double gap2, double W2, double We2, double
L2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discSymInvPosLCAPuIV
public static void createDiscSymInvPosLCAPuIV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double Lc, double W, double We, double Ls, double gap2, double
Lc2, double W2, double We2, double L2, boolean ENDCAP, int gdsLayer
, int dataType, double shapeReso)

// discSymInvPosLCAPuIV
public static GArea createDiscSymInvPosLCAPuIV(double x, double y,
double radius, double Wr, double re, double gap, double Lc, double
W, double We, double Ls, double gap2, double Lc2, double W2, double
We2, double L2, boolean ENDCAP, int gdsLayer, int dataType, double
shapeReso)

// ringSymInvPosLCAPuIV
public static void createRingSymInvPosLCAPuIV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap

```

```

, double Lc, double W, double We, double Ls, double gap2, double
Lc2, double W2, double We2, double L2, boolean ENDCAP, int gdsLayer
, int dataType, double shapeReso)

// ringSymInvPosLCAPulV
public static GArea createRingSymInvPosLCAPulV(double x, double y,
double radius, double Wr, double re, double gap, double Lc, double
W, double We, double Ls, double gap2, double Lc2, double W2, double
We2, double L2, boolean ENDCAP, int gdsLayer, int dataType, double
shapeReso)

// discPulleyAV
public static void createDiscPulleyAV(Struct currentStruct, double x,
double y, double radius, double Wr, double gap, double angle,
double W, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// discPulleyAV
public static GArea createDiscPulleyAV(double x, double y, double
radius, double Wr, double gap, double angle, double W, double Ls,
boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPulleyAV
public static void createRingPulleyAV(Struct currentStruct, double x,
double y, double radius, double Wr, double gap, double angle,
double W, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// ringPulleyAV
public static GArea createRingPulleyAV(double x, double y, double
radius, double Wr, double gap, double angle, double W, double Ls,
boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulADSV
public static void createDiscPulADSV(Struct currentStruct, double x,
double y, double radius, double Wr, double gap, double angle,
double W, double Ls, double gap2, double W2, double L2, boolean
ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulADSV
public static GArea createDiscPulADSV(double x, double y, double radius
, double Wr, double gap, double angle, double W, double Ls, double
gap2, double W2, double L2, boolean ENDCAP, int gdsLayer, int
dataType, double shapeReso)

// ringPulADSV
public static void createRingPulADSV(Struct currentStruct, double x,
double y, double radius, double Wr, double gap, double angle,
double W, double Ls, double gap2, double W2, double L2, boolean
ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPulADSV
public static GArea createRingPulADSV(double x, double y, double radius
, double Wr, double gap, double angle, double W, double Ls, double
gap2, double W2, double L2, boolean ENDCAP, int gdsLayer, int
dataType, double shapeReso)

```

```

// discPulAPulV
public static void createDiscPulAPulV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double Ls, double gap2, double angle2, double W2, double
    Ls2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulAPulV
public static GArea createDiscPulAPulV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double Ls,
    double gap2, double angle2, double W2, double Ls2, boolean ENDCAP,
    int gdsLayer, int dataType, double shapeReso)

// ringPulAPulV
public static void createRingPulAPulV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double Ls, double gap2, double angle2, double W2, double
    Ls2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPulAPulV
public static GArea createRingPulAPulV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double Ls,
    double gap2, double angle2, double W2, double Ls2, boolean ENDCAP,
    int gdsLayer, int dataType, double shapeReso)

// discPulleyLCAV
public static void createDiscPulleyLCAV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double
    W, double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discPulleyLCAV
public static GArea createDiscPulleyLCAV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPulleyLCAV
public static void createRingPulleyLCAV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double
    W, double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringPulleyLCAV
public static GArea createRingPulleyLCAV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulLCADSV
public static void createDiscPulLCADSV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double W
    , double Ls, double gap2, double W2, double L2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// discPulLCADSV
public static GArea createDiscPulLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double W2, double L2, boolean ENDCAP, int gdsLayer,
    int dataType, double shapeReso)

```



```

// ringPulLCADSV
public static void createRingPulLCADSV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double W
    , double Ls, double gap2, double W2, double L2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// ringPulLCADSV
public static GArea createRingPulLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double W2, double L2, boolean ENDCAP, int gdsLayer,
    int dataType, double shapeReso)

// discPulLCAPuV
public static void createDiscPulLCAPuV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double
    W, double Ls, double gap2, double Lc2, double W2, double Ls2,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulLCAPuV
public static GArea createDiscPulLCAPuV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double Lc2, double W2, double Ls2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// ringPulLCAPuV
public static void createRingPulLCAPuV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double Lc, double
    W, double Ls, double gap2, double Lc2, double W2, double Ls2,
    boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPulLCAPuV
public static GArea createRingPulLCAPuV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double Ls,
    double gap2, double Lc2, double W2, double Ls2, boolean ENDCAP, int
    gdsLayer, int dataType, double shapeReso)

// discPulleyInvAV
public static void createDiscPulleyInvAV(Struct currentStruct, double x
    , double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discPulleyInvAV
public static GArea createDiscPulleyInvAV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringPulleyInvAV
public static void createRingPulleyInvAV(Struct currentStruct, double x
    , double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringPulleyInvAV

```

```

public static GArea createRingPulleyInvAV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discPulInvADSV
public static void createDiscPulInvADSV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discPulInvADSV
public static GArea createDiscPulInvADSV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPulInvADSV
public static void createRingPulInvADSV(Struct currentStruct, double x,
    double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringPulInvADSV
public static GArea createRingPulInvADSV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulInvAPuLV
public static void createDiscPulInvAPuLV(Struct currentStruct, double x
    , double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double angle2, double
    W2, double We2, double Ls2, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discPulInvAPuLV
public static GArea createDiscPulInvAPuLV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double angle2, double W2, double We2,
    double Ls2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringPulInvAPuLV
public static void createRingPulInvAPuLV(Struct currentStruct, double x
    , double y, double radius, double Wr, double gap, double angle,
    double W, double We, double Ls, double gap2, double angle2, double
    W2, double We2, double Ls2, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringPulInvAPuLV
public static GArea createRingPulInvAPuLV(double x, double y, double
    radius, double Wr, double gap, double angle, double W, double We,
    double Ls, double gap2, double angle2, double W2, double We2,
    double Ls2, boolean ENDCAP, int gdsLayer, int dataType, double

```

```

        shapeReso)

// discPulleyInvLCAV
public static void createDiscPulleyInvLCAV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// discPulleyInvLCAV
public static GArea createDiscPulleyInvLCAV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringPulleyInvLCAV
public static void createRingPulleyInvLCAV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, boolean ENDCAP, int gdsLayer, int
    dataType, double shapeReso)

// ringPulleyInvLCAV
public static GArea createRingPulleyInvLCAV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discPullInvLCADSV
public static void createDiscPullInvLCADSV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// discPullInvLCADSV
public static GArea createDiscPullInvLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPullInvLCADSV
public static void createRingPullInvLCADSV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, double gap2, double W2, double We2,
    double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

// ringPullInvLCADSV
public static GArea createRingPullInvLCADSV(double x, double y, double
    radius, double Wr, double gap, double Lc, double W, double We,
    double Ls, double gap2, double W2, double We2, double L2, boolean
    ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPullInvLCAPuV
public static void createDiscPullInvLCAPuV(Struct currentStruct, double
    x, double y, double radius, double Wr, double gap, double Lc,
    double W, double We, double Ls, double gap2, double Lc2, double W2,
    double We2, double Ls2, boolean ENDCAP, int gdsLayer, int dataType

```

```

        , double shapeReso)

// discPullInvLCAPuV
public static GArea createDiscPullInvLCAPuV(double x, double y, double
radius, double Wr, double gap, double Lc, double W, double We,
double Ls, double gap2, double Lc2, double W2, double We2, double
Ls2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPullInvLCAPuV
public static void createRingPullInvLCAPuV(Struct currentStruct, double
x, double y, double radius, double Wr, double gap, double Lc,
double W, double We, double Ls, double gap2, double Lc2, double W2,
double We2, double Ls2, boolean ENDCAP, int gdsLayer, int dataType
, double shapeReso)

// ringPullInvLCAPuV
public static GArea createRingPullInvLCAPuV(double x, double y, double
radius, double Wr, double gap, double Lc, double W, double We,
double Ls, double gap2, double Lc2, double W2, double We2, double
Ls2, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulleyInvPosAV
public static void createDiscPulleyInvPosAV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double angle, double W, double We, double Ls, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// discPulleyInvPosAV
public static GArea createDiscPulleyInvPosAV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// ringPulleyInvPosAV
public static void createRingPulleyInvPosAV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double angle, double W, double We, double Ls, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// ringPulleyInvPosAV
public static GArea createRingPulleyInvPosAV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// discPullInvPADSV
public static void createDiscPullInvPADSV(Struct currentStruct, double x
, double y, double radius, double Wr, double re, double gap, double
angle, double W, double We, double Ls, double gap2, double W2,
double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// discPullInvPADSV
public static GArea createDiscPullInvPADSV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double W2, double We2, double L2
, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

```

```

// ringPullInvPADSV
public static void createRingPullInvPADSV(Struct currentStruct, double x
, double y, double radius, double Wr, double re, double gap, double
angle, double W, double We, double Ls, double gap2, double W2,
double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
double shapeReso)

// ringPullInvPADSV
public static GArea createRingPullInvPADSV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double W2, double We2, double L2
, boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPullInvPAPuLV
public static void createDiscPullInvPAPuLV(Struct currentStruct, double
x, double y, double radius, double Wr, double re, double gap,
double angle, double W, double We, double Ls, double gap2, double
angle2, double W2, double We2, double L2, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// discPullInvPAPuLV
public static GArea createDiscPullInvPAPuLV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double angle2, double W2, double
We2, double L2, boolean ENDCAP, int gdsLayer, int dataType, double
shapeReso)

// ringPullInvPAPuLV
public static void createRingPullInvPAPuLV(Struct currentStruct, double
x, double y, double radius, double Wr, double re, double gap,
double angle, double W, double We, double Ls, double gap2, double
angle2, double W2, double We2, double L2, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// ringPullInvPAPuLV
public static GArea createRingPullInvPAPuLV(double x, double y, double
radius, double Wr, double re, double gap, double angle, double W,
double We, double Ls, double gap2, double angle2, double W2, double
We2, double L2, boolean ENDCAP, int gdsLayer, int dataType, double
shapeReso)

// discPulleyInvPosLCAV
public static void createDiscPulleyInvPosLCAV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap
, double Lc, double W, double We, double Ls, boolean ENDCAP, int
gdsLayer, int dataType, double shapeReso)

// discPulleyInvPosLCAV
public static GArea createDiscPulleyInvPosLCAV(double x, double y,
double radius, double Wr, double re, double gap, double Lc, double
W, double We, double Ls, boolean ENDCAP, int gdsLayer, int dataType
, double shapeReso)

// ringPulleyInvPosLCAV
public static void createRingPulleyInvPosLCAV(Struct currentStruct,
double x, double y, double radius, double Wr, double re, double gap

```

```

        , double Lc, double W, double We, double Ls, boolean ENDCAP, int
        gdsLayer, int dataType, double shapeReso)

// ringPulleyInvPosLCAV
public static GArea createRingPulleyInvPosLCAV(double x, double y,
        double radius, double Wr, double re, double gap, double Lc, double
        W, double We, double Ls, boolean ENDCAP, int gdsLayer, int dataType
        , double shapeReso)

// discPulInvPLCADSV
public static void createDiscPulInvPLCADSV(Struct currentStruct, double
        x, double y, double radius, double Wr, double re, double gap,
        double Lc, double W, double We, double Ls, double gap2, double W2,
        double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
        double shapeReso)

// discPulInvPLCADSV
public static GArea createDiscPulInvPLCADSV(double x, double y, double
        radius, double Wr, double re, double gap, double Lc, double W,
        double We, double Ls, double gap2, double W2, double We2, double L2
        , boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// ringPulInvPLCADSV
public static void createRingPulInvPLCADSV(Struct currentStruct, double
        x, double y, double radius, double Wr, double re, double gap,
        double Lc, double W, double We, double Ls, double gap2, double W2,
        double We2, double L2, boolean ENDCAP, int gdsLayer, int dataType,
        double shapeReso)

// ringPulInvPLCADSV
public static GArea createRingPulInvPLCADSV(double x, double y, double
        radius, double Wr, double re, double gap, double Lc, double W,
        double We, double Ls, double gap2, double W2, double We2, double L2
        , boolean ENDCAP, int gdsLayer, int dataType, double shapeReso)

// discPulInvPLCAPuIV
public static void createDiscPulInvPLCAPuIV(Struct currentStruct,
        double x, double y, double radius, double Wr, double re, double gap
        , double Lc, double W, double We, double Ls, double gap2, double
        Lc2, double W2, double We2, double L2, boolean ENDCAP, int gdsLayer
        , int dataType, double shapeReso)

// discPulInvPLCAPuIV
public static GArea createDiscPulInvPLCAPuIV(double x, double y, double
        radius, double Wr, double re, double gap, double Lc, double W,
        double We, double Ls, double gap2, double Lc2, double W2, double
        We2, double L2, boolean ENDCAP, int gdsLayer, int dataType, double
        shapeReso)

// ringPulInvPLCAPuIV
public static void createRingPulInvPLCAPuIV(Struct currentStruct,
        double x, double y, double radius, double Wr, double re, double gap
        , double Lc, double W, double We, double Ls, double gap2, double
        Lc2, double W2, double We2, double L2, boolean ENDCAP, int gdsLayer
        , int dataType, double shapeReso)

// ringPulInvPLCAPuIV

```

```

public static GArea createRingPullInvPLCAPulV(double x, double y, double
    radius, double Wr, double re, double gap, double Lc, double W,
    double We, double Ls, double gap2, double Lc2, double W2, double
    We2, double L2, boolean ENDCAP, int gdsLayer, int dataType, double
    shapeReso)

//
// Alignment Elements
//
// alignBB
public static ArrayList<GArea> createAlignBB(double x, double y, double
    wa1, double wa2, double wb, int La, int Lb, double THETA)

// alignBB
public static void createAlignBB(Struct currentStruct, double x, double
    y, double wa1, double wa2, double wb, int La, int Lb, double
    THETA)

// axes
public static GArea createAxes(double x, double y, double W, double L,
    double arrowWa, double arrowL, double THETA, double shapeReso, int
    gdsLayer)

// axesV2
public static GArea createAxesV2(double x, double y, double W1, double
    L1, double W2, double L2, double arrowWa, double arrowL, double
    THETA, double shapeReso, int gdsLayer)

// axesOutline
public static GArea createAxesOutline(double x, double y, double W,
    double L, double arrowWa, double arrowL, double shapeOutlineWidth,
    double THETA, double shapeReso, int gdsLayer)

// axesOutlineV2
public static GArea createAxesOutlineV2(double x, double y, double W1,
    double L1, double W2, double L2, double arrowWa, double arrowL,
    double shapeOutlineWidth, double THETA, double shapeReso, int
    gdsLayer)

// axesDashed
public static GArea createAxesDashed(double x, double y, double W,
    double L, double arrowWa, double arrowL, double shapeOutlineWidth,
    double dashLength, double dashGap, int CAP, int JOIN, double THETA,
    double shapeReso, int gdsLayer)

// axesDashedV2
public static GArea createAxesDashedV2(double x, double y, double W1,
    double L1, double W2, double L2, double arrowWa, double arrowL,
    double shapeOutlineWidth, double dashLength, double dashGap, int
    CAP, int JOIN, double THETA, double shapeReso, int gdsLayer)

// arrowHeadOutline
public static GArea createArrowHeadOutline(double x, double y, double
    arrowW, double arrowL, double width, double shapeOutlineWidth,
    double THETA, int gdsLayer)

// arrowOutline

```

```

public static GArea createArrowOutline(double x, double y, double
    arrowW, double arrowL, double width, double length, double
    shapeOutlineWidth, double THETA, int gdsLayer)

// arrowArrayOutline
public static GArea createArrowLinearArrayOutline(double x, double y,
    double arrowW, double arrowL, double width, double length, int
    numberOfArrows, double shapeOutlineWidth, double THETA, int
    gdsLayer)

// arrowHeadDashed
public static GArea createArrowHeadDashed(double x, double y, double
    arrowW, double arrowL, double width, double shapeOutlineWidth,
    double dashLength, double dashGap, int CAP, int JOIN, double THETA,
    int gdsLayer)

// arrowDashed
public static GArea createArrowDashed(double x, double y, double arrowW
    , double arrowL, double width, double length, double
    shapeOutlineWidth, double dashLength, double dashGap, int CAP, int
    JOIN, double THETA, int gdsLayer)

// arrowArrayDashed
public static GArea createArrowLinearArrayDashed(double x, double y,
    double arrowW, double arrowL, double width, double length, int
    numberOfArrows, double shapeOutlineWidth, double dashLength, double
    dashGap, int CAP, int JOIN, double THETA, int gdsLayer)

//
// Arrays and Instances
//
// arrayRectC
public static void createArrayRectC(Struct structToBeArrayed, Struct
    currentStruct, double xC, double yC, int numColumns, int numRows,
    double dx, double dy)

// arrayHexC
public static void createArrayHexC(Struct structToBeArrayed, Struct
    currentStruct, double xC, double yC, int numColumns, int numRows,
    double ds)

//
// Objects
//
// arcRadialFill
public static void createArcRadialFill(Struct currentStruct, String
    uniqueStructName, double x, double y, double radTorus, double
    widthTorus, double circleRad, int numSides, double pitch, double
    edgeDistance, double thetaStart, double thetaEnd, int gdsLayer)

// radialCircle
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y r0 N0 R1 r1 dr1 N1 R2 r2 dr2 N2 .... Rn rn drn Nn
public static void createRadialCircle(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer)

```



```

// radialCircleV
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y r0 R1 r1 dr1 R2 r2 dr2 .... Rn rn drn
public static void createRadialCircleV(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer, double
    shapeReso)

// radialCircleU
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y r0 NO R1 r1 dr1 N1 R2 r2 dr2 N2 .... Rn rn drn Nn
public static void createRadialCircleU(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer)

// radialCircleUV
// ArrayList<Double> al is defined as (Note: integer N's are included):
// uniqueStructName x y r0 R1 r1 dr1 R2 r2 dr2 .... Rn rn drn
public static void createRadialCircleUV(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer, double
    shapeReso)

// radialCircle2
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y r0 NO LO DO THETA0 R1 r1 dr1 N1 L1 D1 THETA1 .... Rn rn drn Nn
// Ln Dn THETAn
public static void createRadialCircle2(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer)

// radialCircleV2
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y r0 LO DO THETA0 R1 r1 dr1 L1 D1 THETA1 .... Rn rn drn Ln Dn
// THETAn
public static void createRadialCircleV2(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer, double
    shapeReso)

// radialCircleU2
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y r0 NO LO DO THETA0 R1 r1 dr1 N1 L1 D1 THETA1 .... Rn rn drn Nn
// Ln Dn THETAn
public static void createRadialCircleU2(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer)

// radialCircleUV2
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y r0 LO DO THETA0 R1 r1 dr1 L1 D1 THETA1 .... Rn rn drn Ln Dn
// THETAn
public static void createRadialCircleUV2(Struct currentStruct, String
    radialCircleName, ArrayList<Double> al, int gdsLayer, double
    shapeReso)

// random Radial
public static void createRandomRadial(Struct currentStruct, double x,
    double y, double radMin, double radMax, double
    potentialBetweenElements, int numberOfElements, int
    numberOfIterations, double radiusBIGcircle, int numSidesSmall, int
    numSidesBIGcircle, int gdsLayerMin, int gdsLayerMax, int
    gdsLayerBIGcircle) throws IOException

```

```

// random Radial SVG
public static void createRandomRadialSVG(Struct currentStruct, double x
, double y, double radMin, double radMax, double
potentialBetweenElements, int numberOfElements, int
numberOfIterations, double radiusBIGcircle, int numSidesSmall, int
numSidesBIGcircle, int gdsLayerMin, int gdsLayerMax, int
gdsLayerBIGcircle, double svgWallThickness, String svgFileName,
boolean SVGOUTPUT) throws IOException

// spiralArchST – Archimedes spiral with skipped turns
public static ArrayList<GArea> createSpiralArchST(double x, double y,
double width, int numberOfTurns, double separationBetweenTurns, int
pointsPerTurn, int skippedTurns, double endLength, double THETA,
int ENDCAP, int gdsLayer)

// spiralArchST – Archimedes spiral with skipped turns
public static void createSpiralArchST(Struct currentStruct, double x,
double y, double width, int numberOfTurns, double
separationBetweenTurns, int pointsPerTurn, int skippedTurns, double
endLength, double THETA, int ENDCAP, int gdsLayer)

// crossWiresV1
public static ArrayList<GArea> createCrossWiresV1(double x, double y,
double w1, double w2, double p1, double p2, double L1, double L2,
double L3, double a, double b, int numPaths, int Lyr1, int Lyr2,
int Lyr3)

// crossWiresV1
public static void createCrossWiresV1(Struct currentStruct, double x,
double y, double w1, double w2, double p1, double p2, double L1,
double L2, double L3, double a, double b, int numPaths, int Lyr1,
int Lyr2, int Lyr3)

// crossWiresV2
public static ArrayList<GArea> createCrossWiresV2(double x, double y,
double w1, double w2, double p1, double p2, double L1, double L2,
double L3, double a, double b, int numPaths, int Lyr1, int Lyr2,
int Lyr3)

// crossWiresV2
public static void createCrossWiresV2(Struct currentStruct, double x,
double y, double w1, double w2, double p1, double p2, double L1,
double L2, double L3, double a, double b, int numPaths, int Lyr1,
int Lyr2, int Lyr3)

// crossWiresV3
public static ArrayList<GArea> createCrossWiresV3(double x, double y,
double w1, double w2, double p1, double p2, double L1, double L2,
double a, double b, int numPaths, int Lyr1, int Lyr2, int Lyr3)

// crossWiresV3
public static void createCrossWiresV3(Struct currentStruct, double x,
double y, double w1, double w2, double p1, double p2, double L1,
double L2, double a, double b, int numPaths, int Lyr1, int Lyr2,
int Lyr3)

```

```

// torusEllipseW
public static GArea createTorusEllipseW(double x, double y, double
    radiusX, double radiusY, double width, double angleStart, double
    angleEnd, int numPoints, double THETA, int gdsLayer)

// torusEllipseFocusW
public static GArea createTorusEllipseFocusW(double x, double y, double
    radiusX, double radiusY, double width, double angleStart, double
    angleEnd, int numPoints, double THETA, int gdsLayer)

// microfluidic cross junctions V1
public static GArea createCrossJunctionV1(double x, double y, double d,
    double w, double L1, double L2, double r, int numSides, double
    THETA, int gdsLayer)

// microfluidic cross junctions V1
public static void createCrossJunctionV1(Struct currentStruct, double x
    , double y, double d, double w, double L1, double L2, double r, int
    numSides, double THETA, int gdsLayer)

// microfluidic cross junctions V2
public static GArea createCrossJunctionV2(double x, double y, double d,
    double w, double L1, double L2, double L3, double L4, double r,
    int numSides, double THETA, int gdsLayer)

// microfluidic cross junctions V2
public static void createCrossJunctionV2(Struct currentStruct, double x
    , double y, double d, double w, double L1, double L2, double L3,
    double L4, double r, int numSides, double THETA, int gdsLayer)

// microfluidic cross junctions V3
public static ArrayList<GArea> createCrossJunctionV3(double x, double y
    , double d, double w, double L1, double L2, double L3, double L4,
    int Lyr1, int Lyr2, double r, double dr, int numSides, double THETA
    )

// microfluidic cross junctions V3
public static void createCrossJunctionV3(Struct currentStruct, double x
    , double y, double d, double w, double L1, double L2, double L3,
    double L4, int Lyr1, int Lyr2, double r, double dr, int numSides,
    double THETA)

// funnel
public static GArea createFunnel(double x, double y, double w1a, double
    L1a, double w1b, double L1b, double THETA, int gdsLayer, double
    shapeReso)

// funnel
public static void createFunnel(Struct currentStruct, double x, double
    y, double w1a, double L1a, double w1b, double L1b, double THETA,
    int gdsLayer, double shapeReso)

// funnelR
public static GArea createFunnelR(double x, double y, double w1a,
    double L1a, double w1b, double L1b, double L1r, double THETA, int
    gdsLayer, double shapeReso)

```

```

// funnelR
public static void createFunnelR(Struct currentStruct, double x, double
    y, double w1a, double L1a, double w1b, double L1b, double L1r,
    double THETA, int gdsLayer, double shapeReso)

// funnelJunctionV1
public static GArea createFunnelJunctionV1(double x, double y, double
    w1a, double w1b, double L1a, double L1b, double w2a, double w2b,
    double L2a, double L2b, double w3a, double w3b, double L3a, double
    L3b, double w4a, double w4b, double L4a, double L4b, double THETA,
    int gdsLayer, double shapeReso)

// funnelJunctionV1
public static void createFunnelJunctionV1(Struct currentStruct, double
    x, double y, double w1a, double w1b, double L1a, double L1b, double
    w2a, double w2b, double L2a, double L2b, double w3a, double w3b,
    double L3a, double L3b, double w4a, double w4b, double L4a, double
    L4b, double THETA, int gdsLayer, double shapeReso)

// funnelJunctionV1R
public static GArea createFunnelJunctionV1R(double x, double y, double
    w1a, double w1b, double L1a, double L1b, double L1r, double w2a,
    double w2b, double L2a, double L2b, double L2r, double w3a, double
    w3b, double L3a, double L3b, double L3r, double w4a, double w4b,
    double L4a, double L4b, double L4r, double THETA, int gdsLayer,
    double shapeReso)

// funnelJunctionV1R
public static void createFunnelJunctionV1R(Struct currentStruct, double
    x, double y, double w1a, double w1b, double L1a, double L1b,
    double L1r, double w2a, double w2b, double L2a, double L2b, double
    L2r, double w3a, double w3b, double L3a, double L3b, double L3r,
    double w4a, double w4b, double L4a, double L4b, double L4r, double
    THETA, int gdsLayer, double shapeReso)

// funnelJunctionV2
public static GArea createFunnelJunctionV2(double x, double y, double
    w1, double L1a, double L1b, double w2, double L2, double w3, double
    L3a, double L3b, double THETA, int gdsLayer, double shapeReso)

// funnelJunctionV2
public static void createFunnelJunctionV2(Struct currentStruct, double
    x, double y, double w1, double L1a, double L1b, double w2, double
    L2, double w3, double L3a, double L3b, double THETA, int gdsLayer,
    double shapeReso)

// funnelJunctionV2C
public static GArea createFunnelJunctionV2C(double x, double y, double
    w1, double L1a, double L1b, double w2, double L2, double w3, double
    L3a, double L3b, double THETA, int gdsLayer, double shapeReso)

// funnelJunctionV2C
public static void createFunnelJunctionV2C(Struct currentStruct, double
    x, double y, double w1, double L1a, double L1b, double w2, double
    L2, double w3, double L3a, double L3b, double THETA, int gdsLayer,
    double shapeReso)

```

```

// splinePath
public static ArrayList<GArea> createSplinePath(ArrayList<Double>
    pointsXY, int numberOfSplinePoints, double width, int CAP, int JOIN
    , int numberOfFractureShapes, int gdsLayer)

// splinePath
public static void createSplinePath(Struct currentStruct, ArrayList<
    Double> pointsXY, double numberOfSplinePoints, double width, int
    CAP, int JOIN, int numberOfFractureShapes, int gdsLayer)

// splineClosedPath
public static ArrayList<GArea> createSplineClosedPath(ArrayList<Double>
    pointsXY, int numberOfSplinePoints, double width, int CAP, int
    JOIN, int numberOfFractureShapes, int gdsLayer)

// splineClosedPath
public static void createSplineClosedPath(Struct currentStruct,
    ArrayList<Double> pointsXY, double numberOfSplinePoints, double
    width, int CAP, int JOIN, int numberOfFractureShapes, int gdsLayer)

// splineShape
public static ArrayList<GArea> createSplineShape(ArrayList<Double>
    pointsXY, int numberOfSplinePoints, int CAP, int JOIN, int
    numberOfFractureShapes, int gdsLayer)

// splineShape
public static void createSplineShape(Struct currentStruct, ArrayList<
    Double> pointsXY, double numberOfSplinePoints, int CAP, int JOIN,
    int numberOfFractureShapes, int gdsLayer)

// roundPath
// ArrayList<Double> pointsXYRN – x1 y1 x2 y2 r2 N2 x3 y3 r3 N3 x4 y4
// r4 N4 .... xn yn
public static ArrayList<GArea> createRoundPath(ArrayList<Double>
    pointsXYRN, double width, int CAP, int JOIN, int
    numberOfFractureShapes, int gdsLayer)

// roundPath
// ArrayList<Double> pointsXYRN – x1 y1 x2 y2 r2 N2 x3 y3 r3 N3 x4 y4
// r4 N4 .... xn yn
public static void createRoundPath(Struct currentStruct, ArrayList<
    Double> pointsXY, double width, int CAP, int JOIN, int
    numberOfFractureShapes, int gdsLayer)

// roundClosedPath
// ArrayList<Double> pointsXYRN – x1 y1 x2 y2 r2 N2 x3 y3 r3 N3 x4 y4
// r4 N4 .... xn yn
public static ArrayList<GArea> createRoundClosedPath(ArrayList<Double>
    pointsXYRN, double width, int CAP, int JOIN, int
    numberOfFractureShapes, int gdsLayer)

// roundClosedPath
// ArrayList<Double> pointsXYRN – x1 y1 x2 y2 r2 N2 x3 y3 r3 N3 x4 y4
// r4 N4 .... xn yn
public static void createRoundClosedPath(Struct currentStruct,
    ArrayList<Double> pointsXY, double width, int CAP, int JOIN, int
    numberOfFractureShapes, int gdsLayer)

```

```

// roundShape
// ArrayList<Double> pointsXYRN - x1 y1 x2 y2 r2 N2 x3 y3 r3 N3 x4 y4
//   r4 N4 .... xn yn
public static ArrayList<GArea> createRoundShape(ArrayList<Double>
    pointsXYRN, int CAP, int JOIN, int numberOfFractureShapes, int
    gdsLayer)

// roundShape
// ArrayList<Double> pointsXYRN - x1 y1 x2 y2 r2 N2 x3 y3 r3 N3 x4 y4
//   r4 N4 .... xn yn
public static void createRoundShape(Struct currentStruct, ArrayList<
    Double> pointsXY, int CAP, int JOIN, int numberOfFractureShapes,
    int gdsLayer)

// resoPatternLS2
public static GArea createResoPatternLS2(double x, double y, double w,
    double L1, double L2, double p1, double p2, int numElements, int
    gdsLayer)

// resoPatternLS2
public static void createResoPatternLS2(Struct currentStruct, double x,
    double y, double w, double L1, double L2, double p1, double p2,
    int numElements, int gdsLayer)

// spinIceA
public static void createSpinIceA(Struct currentStruct, String
    uniqueStructName, double x, double y, double radiusX, double
    radiusY, double space, double percentEmpty, int numSides, int
    numElementsX, int numElementsY, int gdsLayer)

// spinIceVectorA
public static void createSpinIceVectorA(Struct currentStruct, String
    uniqueStructName, double x, double y, double radiusX, double
    radiusY, double space, double percentEmpty, int numElementsX, int
    numElementsY, double shapeReso, int gdsLayer)

// spinIceB
public static void createSpinIceB(Struct currentStruct, String
    uniqueStructName, double x, double y, double L, double H, double
    space, double percentEmpty, int numElementsX, int numElementsY,
    double shapeReso, int gdsLayer)

// Van der Pauw version 1
public static ArrayList<GArea> createVanDerPauwV1(double x, double y,
    double d, double w, double L1, double L2, double a, double b, int
    LyrA, int LyrB)

// Van der Pauw version 1
public static void createVanDerPauwV1(Struct currentStruct, double x,
    double y, double d, double w, double L1, double L2, double a,
    double b, int LyrA, int LyrB)

// Van der Pauw version 2
public static ArrayList<GArea> createVanDerPauwV2(double x, double y,
    double d, double w, double L1, double L2, double L3, double L4,
    double a, double b, int LyrA, int LyrB)

```

```

// Van der Pauw version 2
public static void createVanDerPauwV2(Struct currentStruct, double x,
    double y, double d, double w, double L1, double L2, double L3,
    double L4, double a, double b, int LyrA, int LyrB)

// wire contacts - wireC1
public static ArrayList<GArea> createWireC1(double x, double y, double
    w, double h1, double h2, double h3, int LyrA, int LyrB, int LyrC)

// wire contacts - wireC1
public static void createWireC1(Struct currentStruct, double x, double
    y, double w, double h1, double h2, double h3, int LyrA, int LyrB,
    int LyrC)

// beamCurvedEnds
public static GArea createBeamCurvedEnds(double x, double y, double L,
    double W, double r, int numSides, double THETA, int gdsLayer)

// beamCurvedEndsV
public static GArea createBeamCurvedEndsV(double x, double y, double L,
    double W, double r, double THETA, int gdsLayer, double shapeReso)

// beamCurvedEndsC
public static GArea createBeamCurvedEndsC(double x, double y, double L,
    double W, double r, int numSides, double THETA, int gdsLayer)

// beamCurvedEndsCV
public static GArea createBeamCurvedEndsCV(double x, double y, double L
    , double W, double r, double THETA, int gdsLayer, double shapeReso)

// bolometerL2
public static ArrayList<GArea> createBolometerL2(double x, double y,
    double w1, double w2, double w3, double g1, double g2, double g3,
    double g4, double L1, double L2, double r1, double r2, double r,
    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)

// bolometerL2
public static void createBolometerL2(Struct currentStruct, double x,
    double y, double w1, double w2, double w3, double g1, double g2,
    double g3, double g4, double L1, double L2, double r1, double r2,
    double r, 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)

// suspended ring array - MEMS Rings Version 1
public static ArrayList<GArea> createMemsRingsV1(double x, double y,
    double radInner, double ringWidth, double ringPitch, int numRings,
    double postW, double squareExtent, double THETA, int gdsLayer,
    double shapeReso)

// suspended ring array - MEMS Rings Version 1
public static void createMemsRingsV1(Struct currentStruct, double x,
    double y, double radInner, double ringWidth, double ringPitch, int
    numRings, double postW, double squareExtent, double THETA, int

```

```

        gdsLayer, double shapeReso)

// suspended ring array – MEMS Rings Version 2
public static ArrayList<GArea> createMemsRingsV2(double x, double y,
        double radInner, double ringWidth, double ringPitch, int numRings,
        double postW, double squareExtent, double THETA, int gdsLayer,
        double shapeReso)

// suspended ring array – MEMS Rings Version 2
public static void createMemsRingsV2(Struct currentStruct, double x,
        double y, double radInner, double ringWidth, double ringPitch, int
        numRings, double postW, double squareExtent, double THETA, int
        gdsLayer, double shapeReso)

// suspended ring array – MEMS Rings Version 3
public static ArrayList<GArea> createMemsRingsV3(double x, double y,
        double radInner, double ringWidth, double ringPitch, int numRings,
        double postW, double squareExtent, double THETA, int gdsLayer,
        double shapeReso)

// suspended ring array – MEMS Rings Version 3
public static void createMemsRingsV3(Struct currentStruct, double x,
        double y, double radInner, double ringWidth, double ringPitch, int
        numRings, double postW, double squareExtent, double THETA, int
        gdsLayer, double shapeReso)

// *****
//
// Miscellaneous GDS Area and Struct Methods
//
// *****

// GDS Area translate
public static GArea createGAreaTranslate (GArea ga, double x, double y)

// GDS Area rotate (in degrees) about point x,y
public static GArea createGAreaRotate (GArea ga, double x, double y,
        double THETA)

// mirroring GArea ga around x-axis
public static GArea createGAreaMirrorX(GArea ga)

// mirroring GArea ga around y-axis
public static GArea createGAreaMirrorY(GArea ga)

// mirroring GArea ga around x-axis
public static GArea createMirrorX(GArea ga)

// mirroring GArea ga around y-axis
public static GArea createMirrorY(GArea ga)

// create instance at x,y
public static void createInstance(Struct structToBeInstanced, Struct
        currentStruct, double x, double y)

// create instance x,y and rotation

```



```

public static void createInstance(Struct structToBeInstanced, Struct
    currentStruct, double x, double y, double THETA)

// create instance - Mirror Y
public static void createInstance(Struct structToBeInstanced, Struct
    currentStruct, double x, double y, boolean mirrorY)

// create instance - mirroring occurs around y. mirroring around x -
    set mirrorY = true and THETA = 180
public static void createInstance(Struct structToBeInstanced, Struct
    currentStruct, double x, double y, boolean mirrorY, double THETA)

// create instance - mirroring occurs around y. mirroring around x -
    set mirrorY = true and THETA = 180
public static void createInstance(Struct structToBeInstanced, Struct
    currentStruct, double x, double y, boolean mirrorY, double mag,
    double THETA)

// genAreaOutline - creates a general area outline from GArea ga
public static GArea createGenAreaOutline(GArea ga, double
    shapeOutlineWidth, int gdsLayer, int dataType, double shapeReso)

// genAreaOutline - using globally stored dataType
public static GArea createGenAreaOutline(GArea ga, double
    shapeOutlineWidth, int gdsLayer, double shapeReso)

// genAreaOutline - using globally stored dataType, without shapeReso
public static GArea createGenAreaOutline(GArea ga, double
    shapeOutlineWidth, int gdsLayer)

// createGenAreaOutlineDashed - creates a general area dashed outline
    from GArea ga
public static GArea createGenAreaOutlineDashed(GArea ga, double
    shapeOutlineWidth, double dashLength, double dashGap, int CAP, int
    JOIN, int gdsLayer, int dataType, double shapeReso)

// createGenAreaOutlineDashed - using globally stored dataType
public static GArea createGenAreaOutlineDashed(GArea ga, double
    shapeOutlineWidth, double dashLength, double dashGap, int CAP, int
    JOIN, int gdsLayer, double shapeReso)

// createGenAreaDashed same as createGenAreaOutlineDashed - creates a
    general area dashed outline from GArea ga
public static GArea createGenAreaDashed(GArea ga, double
    shapeOutlineWidth, double dashLength, double dashGap, int CAP, int
    JOIN, int gdsLayer, int dataType, double shapeReso)

// createGenAreaDashed same as createGenAreaOutlineDashed - using
    globally stored dataType
public static GArea createGenAreaDashed(GArea ga, double
    shapeOutlineWidth, double dashLength, double dashGap, int CAP, int
    JOIN, int gdsLayer, double shapeReso)

// *****
//
// Radial Arrays
//

```

```

// *****

// radialArray
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y rx0 ry0 theta0 Nv0 R1 rx1 ry1 theta1 Nv1 p1.... RN rxN ryN
//   thetaN NvN pN
public static void createRadialArray(Struct currentStruct, String
    uniqueStructPrefix, ArrayList<Double> al, int gdsLayer)

// radialArrayW
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y rx0 ry0 theta0 Nv0 R1 rx1 ry1 theta1 Nv1 p1.... RN rxN ryN
//   thetaN NvN pN
public static void createRadialArrayW(Struct currentStruct, String
    uniqueStructPrefix, ArrayList<Double> al, int gdsLayer)

// createRadialArrayConst - constant pitch between rings
public static void createRadialArrayConst(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitch, double ringOuterRadius, int gdsLayer)

// createRadialArrayConstW - constant pitch between rings - full ring
public static void createRadialArrayConstW(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX,
    double shapeRadiusY, double shapeRotation, int numberOfVertices,
    double shapePitch, double ringPitch, double ringOuterRadius, int
    gdsLayer)

// createRadialArrayLin - linear pitch variation between rings
public static void createRadialArrayLin(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitch, double ringPitchDelta, double
    ringOuterRadius, int gdsLayer)

// createRadialArrayLinW - linear pitch variation between rings - full
// ring
public static void createRadialArrayLinW(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitch, double ringPitchDelta, double
    ringOuterRadius, int gdsLayer)

// createRadialCircleArrayLin2 - symmetric linear increase and
// decrease in pitch variation between rings
public static void createRadialArrayLin2(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitch, double ringPitchDelta, int
    numberOfRings, int gdsLayer)

// createRadialArrayLin2W - symmetric linear increase and decrease in
// pitch variation between rings - full ring
public static void createRadialArrayLin2W(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double

```

```

    shapePitch, double ringPitch, double ringPitchDelta, int
    numberOfRings, int gdsLayer)

// radialCircleArrayExp – exponential pitch variation between rings
public static void createRadialArrayExp(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitchStart, double ringPitchEnd, int
    numberOfRings, int gdsLayer)

// radialCircleArrayExpW – exponential pitch variation between rings –
// full ring
public static void createRadialArrayExpW(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitchStart, double ringPitchEnd, int
    numberOfRings, int gdsLayer)

// radialCircleArrayExp2 – symmetric linear increase and decrease in
// pitch variation between rings
public static void createRadialArrayExp2(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitchStart, double ringPitchEnd, int
    numberOfRings, int gdsLayer)

// radialCircleArrayExp2W – symmetric linear increase and decrease in
// pitch variation between rings – full ring
public static void createRadialArrayExp2W(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    shapePitch, double ringPitchStart, double ringPitchEnd, int
    numberOfRings, int gdsLayer)

// *****
//
// Non-uniform arrays (NUA)
//
// *****

// generalized nonuniform array (NUA)
// ArrayList<Double> al is defined as (Note: integer N's are included):
// x y rx0 ry0 theta0 Nv0 dp0 p1 rx1 ry1 theta1 Nv1 dp1 Ns1 .... pN
// rxN ryN thetaN NvN dpN NsN
public static void createNUA(Struct currentStruct, String
    uniqueStructPrefix, ArrayList<Double> al, int gdsLayer)

//
// Linearly Varying Pitch
//

// createNUALin – Linearly varying pitch array
public static void createNUALin(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

```

```

// createNUALinA – Linearly varying pitch array aligned
public static void createNUALinA(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALin2 – Linearly varying pitch symmetric array
public static void createNUALin2(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALinA2 – Linearly varying pitch symmetric, aligned array
public static void createNUALinA2(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALin2D – Linearly varying pitch symmetric, double wide
// array
public static void createNUALin2D(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALinA2D – Linearly varying pitch symmetric, double wide,
// aligned array
public static void createNUALinA2D(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALin3 – Linearly varying vertical and constant horizontal
// pitch array
public static void createNUALin3(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALinA3 – Linearly varying vertical and constant horizontal
// pitch, aligned array
public static void createNUALinA3(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALin4 – Linearly varying vertical and constant horizontal
// pitch, symmetric array
public static void createNUALin4(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALin4D – Linearly varying vertical and constant horizontal
// pitch, symmetric array
public static void createNUALin4D(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double

```

```

        shapeRadiusY, double shapeRotation, int numberOfVertices, double
        pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALinA4 – Linearly varying vertical and constant horizontal
// pitch, aligned, symmetric array
public static void createNUALinA4(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

// createNUALinA4D – Linearly varying vertical and constant horizontal
// pitch, aligned, symmetric array
public static void createNUALinA4D(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchDelta, double arraySize, int gdsLayer)

//
// Linearly Varying Pitch and Radius
//

// createNUALinR – Linearly varying pitch and radius array
public static void createNUALinR(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALinAR – Linearly varying pitch and radius array aligned
public static void createNUALinAR(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALin2R – Linearly varying pitch and radius symmetric array
public static void createNUALin2R(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALinA2R – Linearly varying pitch and radius symmetric,
// aligned array
public static void createNUALinA2R(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALin2DR – Linearly varying pitch and radius symmetric,
// double wide array
public static void createNUALin2DR(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

```

```

// createNUALinA2DR – Linearly varying pitch and radius symmetric,
// double wide, aligned array
public static void createNUALinA2DR(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALin3R – Linearly varying vertical and constant horizontal
// pitch and radius array
public static void createNUALin3R(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX,
    double shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALinA3R – Linearly varying vertical and constant horizontal
// pitch and radius , aligned array
public static void createNUALinA3R(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALin4R – Linearly varying vertical and constant horizontal
// pitch and radius , symmetric array
public static void createNUALin4R(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALinA4R – Linearly varying vertical and constant horizontal
// pitch , aligned , symmetric array
public static void createNUALinA4R(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX,
    double shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALin4DR – Linearly varying vertical and constant horizontal
// pitch and radius , double wide, symmetric array
public static void createNUALin4DR(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

// createNUALinA4DR – Linearly varying vertical and constant horizontal
// pitch and radius , aligned, double wide, symmetric array
public static void createNUALinA4DR(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double deltaRadius, double shapeRotation, int
    numberOfVertices, double pitchStart, double pitchDelta, double
    arraySize, int gdsLayer)

```

```

//
// Exponentially Varying Pitch
//

// createNUAExp – Exponentially varying pitch array
public static void createNUAExp(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

// createNUAExpA – Exponentially varying pitch, aligned array
public static void createNUAExpA(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

// createNUAExp2 – Exponentially varying pitch symmetric array
public static void createNUAExp2(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

// createNUAExp2A – Exponentially varying pitch symmetric, aligned
    array
public static void createNUAExp2A(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

// createNUAExp3 – Exponentially varying vertical and constant
    horizontal pitch array
public static void createNUAExp3(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

// createNUAExp3A – Exponentially varying vertical and constant
    horizontal pitch, symmetric array
public static void createNUAExp3A(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

// createNUAExp4 – Exponentially varying vertical and constant
    horizontal pitch symmetric array
public static void createNUAExp4(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

```

```

// createNUAExp4A – Exponentially varying vertical and constant
// horizontal pitch symmetric array
public static void createNUAExp4A(Struct currentStruct, String
    uniqueStructPrefix, double x, double y, double shapeRadiusX, double
    shapeRadiusY, double shapeRotation, int numberOfVertices, double
    pitchStart, double pitchEnd, double arraySize, int numberOfColumns,
    int gdsLayer)

// *****
//
// Text, Labels, PostScript and Logo Methods
//
// *****

// textFill
public static GArea createTextFill(String textString, String fontName,
    double fontSize, double x, double y, double THETA, int gdsLayer,
    double shapeReso)

// textFillC
public static GArea createTextFillC(String textString, String fontName,
    double fontSize, double x, double y, double THETA, int gdsLayer,
    double shapeReso)

// textOutlineFill
public static GArea createTextOutlineFill(String textString, String
    fontName, double fontSize, double x, double y, double THETA, int
    gdsLayer, double shapeReso)

// textOutlineFillC
public static GArea createTextOutlineFillC(String textString, String
    fontName, double fontSize, double x, double y, double THETA, int
    gdsLayer, double shapeReso)

// textDashed
public static ArrayList<GArea> createTextDashed(Struct currentStruct,
    String textString, String fontName, double fontSize, double x,
    double y, double THETA, double fontOutlineWidth, double dashLength,
    double dashGap, int CAP, int JOIN, int gdsLayer, double shapeReso)

// textDashed
public static void createTextDashed(Struct currentStruct, String
    textString, String fontName, double fontSize, double x, double y,
    double fontOutlineWidth, double dashLength, double dashGap, int CAP
    , int JOIN int gdsLayer, double shapeReso, double THETA)

// textDashedC
public static ArrayList<GArea> createTextDashedC(Struct currentStruct,
    String textString, String fontName, double fontSize, double x,
    double y, double THETA, double fontOutlineWidth, double dashLength,
    double dashGap, int CAP, int JOIN, int gdsLayer, double shapeReso)

// textDashedC
public static void createTextDashedC(Struct currentStruct, String
    textString, String fontName, double fontSize, double x, double y,
    double fontOutlineWidth, double dashLength, double dashGap, int CAP
    , int JOIN, int gdsLayer, double shapeReso, double THETA)

```



```

//
// LABEL FILL
//
// labelFill autoOut
public static void createLabelFillAutoOut(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)

// labelFill autoRowCol
public static void createLabelFillAutoRowCol(int rows, int columns,
    String fontName, double fontSize, double x, double y, double pitchX
    , double pitchY, Struct currentStruct, int gdsLayer, double
    shapeReso)

// labelFill autoOutLett
public static void createLabelFillAutoOutLett(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)

// labelFill autoRowColLett
public static void createLabelFillAutoRowColLett(int rows, int columns,
    String fontName, double fontSize, double x, double y, double
    pitchX, double pitchY, Struct currentStruct, int gdsLayer, double
    shapeReso)

// labelFill CUSTOM OUTER
public static void createLabelFillCustomOuter(String[] stringArray, 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)

// labelFill CUSTOM ROW COLUMN label
public static void createLabelFillCustomRowColumn(String[] stringArray,
    int rows, int columns, String fontName, double fontSize, double x,
    double y, double pitchX, double pitchY, Struct currentStruct, int
    gdsLayer, double shapeReso)

//
// LABEL OUTLINE FILL
//
// labelOutlineFill autoOut
public static void createLabelOutlineFillAutoOut(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)

// labelOutlineFill autoRowCol
public static void createLabelOutlineFillAutoRowCol(int rows, int
    columns, String fontName, double fontSize, double x, double y,
    double pitchX, double pitchY, Struct currentStruct, int gdsLayer,
    double shapeReso)

// labelOutlineFill autoOutLett

```

```

public static void createLabelOutlineFillAutoOutLett(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)

// labelOutlineFill autoRowColLett
public static void createLabelOutlineFillAutoRowColLett(int rows, int
columns, String fontName, double fontSize, double x, double y,
double pitchX, double pitchY, Struct currentStruct, int gdsLayer,
double shapeReso)

// labelOutlineFill CUSTOM OUTER
public static void createLabelOutlineFillCustomOuter(String[]
stringArray, 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)

// labelOutlineFill CUSTOM ROW COLUMN label
public static void createLabelOutlineFillCustomRowColumn(String[]
stringArray, int rows, int columns, String fontName, double
fontSize, double x, double y, double pitchX, double pitchY, Struct
currentStruct, int gdsLayer, double shapeReso)

//
// LABEL DASHED
//
// labelDashed autoOut
public static void createLabelDashedAutoOut(int rows, int columns,
String fontName, double fontSize, double x, double y, double xRow,
double yRow, double pitchX, double pitchY, double dashLength,
double dashGap, int CAP, int JOIN, Struct currentStruct, double
fontOutlineWidth, int gdsLayer, double shapeReso)

// labelDashed autoRowCol
public static void createLabelDashedAutoRowCol(int rows, int columns,
String fontName, double fontSize, double x, double y, double pitchX
, double pitchY, double dashLength, double dashGap, int CAP, int
JOIN, Struct currentStruct, double fontOutlineWidth, int gdsLayer,
double shapeReso)

// labelDashed autoOutLett
public static void createLabelDashedAutoOutLett(int rows, int columns,
String fontName, double fontSize, double x, double y, double xRow,
double yRow, double pitchX, double pitchY, double dashLength,
double dashGap, int CAP, int JOIN, Struct currentStruct, int
gdsLayer, double shapeReso)

// labelDashed autoRowColLett
public static void createLabelDashedAutoRowColLett(int rows, int
columns, String fontName, double fontSize, double x, double y,
double pitchX, double pitchY, double dashLength, double dashGap,
int CAP, int JOIN, Struct currentStruct, int gdsLayer, double
shapeReso)

// labelDashed CUSTOM OUTER

```

```

public static void createLabelDashedCustomOuter(String[] stringArray,
    int rows, int columns, String fontName, double fontSize, double x,
    double y, double xRow, double yRow, double pitchX, double pitchY,
    double dashLength, double dashGap, int CAP, int JOIN, Struct
    currentStruct, int gdsLayer, double shapeReso)

// labelDashed CUSTOM ROW COLUMN label
public static void createLabelDashedCustomRowColumn(String[]
    stringArray, int rows, int columns, String fontName, double
    fontSize, double x, double y, double pitchX, double pitchY, double
    dashLength, double dashGap, int CAP, int JOIN, Struct currentStruct
    , int gdsLayer, double shapeReso)

//
// NIST – CNST logos – Outline and Dashed
//
// cnstEmblemOutline
public static GArea createCnstEmblemLogoOutline(double x, double y,
    double scale, double shapeOutlineWidth, double THETA, double
    shapeReso, int gdsLayer) throws FileNotFoundException

// cnstLogoOutline
public static GArea createCnstLogoOutline(double x, double y, double
    scale, double shapeOutlineWidth, double THETA, double shapeReso,
    int gdsLayer) throws FileNotFoundException

// nistLogoOutline
public static GArea createNistLogoOutline(double x, double y, double
    scale, double shapeOutlineWidth, double THETA, double shapeReso,
    int gdsLayer) throws FileNotFoundException

// nistCnstLogoOutline
public static GArea createNistCnstLogoOutline(double x, double y,
    double scale, double shapeOutlineWidth, double THETA, double
    shapeReso, int gdsLayer) throws FileNotFoundException

// cnstEmblemDashed
public static GArea createCnstEmblemLogoDashed(double x, double y,
    double scale, double shapeOutlineWidth, double dashLength, double
    dashGap, int CAP, int JOIN, double THETA, double shapeReso, int
    gdsLayer) throws FileNotFoundException

// cnstLogoDashed
public static GArea createCnstLogoDashed(double x, double y, double
    scale, double shapeOutlineWidth, double dashLength, double dashGap,
    int CAP, int JOIN, double THETA, double shapeReso, int gdsLayer)
    throws FileNotFoundException

// nistLogoDashed
public static GArea createNistLogoDashed(double x, double y, double
    scale, double shapeOutlineWidth, double dashLength, double dashGap,
    int CAP, int JOIN, double THETA, double shapeReso, int gdsLayer)
    throws FileNotFoundException

// nistCnstLogoDashed
public static GArea createNistCnstLogoDashed(double x, double y, double
    scale, double shapeOutlineWidth, double dashLength, double dashGap

```

```

        , int CAP, int JOIN, double THETA, double shapeReso, int gdsLayer)
        throws FileNotFoundException

//
// PATH LENGTH CALCULATIONS
//
// sets the pathLengthStart
public static void setPathLengthStart(String pathLengthName)

// sets the pathLengthEnd
public static void setPathLengthEnd()

//
// READ AND PARSE GDS FILES
//
// reads GDS file and returns a StringBuffer with parsed GDS file info
public static String readGDS(String fileName, Lib lib) throws
    FileNotFoundException, ScriptException

// reads GDS file and returns a StringBuffer with parsed GDS file info
// - for scripting to include parsed file in logfile
public static String readParseGDS(String fileName, Lib lib) throws
    FileNotFoundException, ScriptException

//
// SHAPES
//
// circle
public static GArea createCircle(double x, double y, double r, int
    numSides, int gdsLayer)

// circleVector
public static GArea createCircleVector(double x, double y, double r,
    int gdsLayer, double shapeReso)

// crossV2
public static GArea createCrossV2(double x, double y, double W1, double
    L1, double W2, double L2, double THETA, int gdsLayer)

// crossRound
public static GArea createCrossRound(double x, double y, double W,
    double L, double r, double THETA, int gdsLayer, double shapeReso)

// crossRoundV2
public static GArea createCrossRoundV2(double x, double y, double W1,
    double L1, double W2, double L2, double r, double THETA, int
    gdsLayer, double shapeReso)

// LshapeRound
public static GArea createLshapeRound(double x, double y, double W1,
    double L1, double W2, double L2, double r, double THETA, int
    gdsLayer, double shapeReso)

// roundSquare
public static GArea createRoundSquare(double xLL, double yLL, double L,
    double r, double THETA, int gdsLayer, double shapeReso)

```

```

// roundSquareC
public static GArea createRoundSquareC(double xC, double yC, double L,
    double r, double THETA, int gdsLayer, double shapeReso)

// square
public static GArea createSquare(double xLL, double yLL, double L,
    double THETA, int gdsLayer)

// squareC
public static GArea createSquareC(double xC, double yC, double L,
    double THETA, int gdsLayer)

// *****
//
// Outline and dashed structures
//
// *****

// circleOutline
public static GArea createCircleOutline(double x, double y, double r,
    double shapeOutlineWidth, int numSides, int gdsLayer)

// circleOutlineVector
public static GArea createCircleOutlineVector(double x, double y,
    double r, double shapeOutlineWidth, int gdsLayer, double shapeReso)

// circleDashed
public static GArea createCircleDashed(double x, double y, double r,
    double shapeOutlineWidth, int numSides, double dashLength, double
    dashGap, int CAP, int JOIN, int gdsLayer, double shapeReso)

// circleDashedVector
public static GArea createCircleDashedVector(double x, double y, double
    r, double shapeOutlineWidth, double dashLength, double dashGap,
    int CAP, int JOIN, int gdsLayer, double shapeReso)

// ellipseOutline
public static GArea createEllipseOutline(double x, double y, double rx,
    double ry, double shapeOutlineWidth, int numSides, double THETA,
    int gdsLayer)

// ellipseOutlineVector
public static GArea createEllipseOutlineVector(double x, double y,
    double rx, double ry, double shapeOutlineWidth, double THETA, int
    gdsLayer, double shapeReso)

// ellipseDashed
public static GArea createEllipseDashed(double x, double y, double rx,
    double ry, double shapeOutlineWidth, int numSides, double
    dashLength, double dashGap, int CAP, int JOIN, double THETA, int
    gdsLayer, double shapeReso)

// ellipseDashedVector
public static GArea createEllipseDashedVector(double x, double y,
    double rx, double ry, double shapeOutlineWidth, double dashLength,
    double dashGap, int CAP, int JOIN, double THETA, int gdsLayer,

```

```

        double shapeReso)

// crossOutline
public static GArea createCrossOutline(double x, double y, double W,
        double L, double shapeOutlineWidth, double THETA, int gdsLayer)

// crossDashed
public static GArea createCrossDashed(double x, double y, double W,
        double L, double shapeOutlineWidth, double dashLength, double
        dashGap, int CAP, int JOIN, double THETA, int gdsLayer, double
        shapeReso)

// crossOutlineV2
public static GArea createCrossOutlineV2(double x, double y, double W1,
        double L1, double W2, double L2, double shapeOutlineWidth, double
        THETA, int gdsLayer)

// crossDashedV2
public static GArea createCrossDashedV2(double x, double y, double W1,
        double L1, double W2, double L2, double shapeOutlineWidth, double
        dashLength, double dashGap, int CAP, int JOIN, double THETA, int
        gdsLayer, double shapeReso)

// crossRoundOutline
public static GArea createCrossRoundOutline(double x, double y, double
        W, double L, double r, double shapeOutlineWidth, double THETA, int
        gdsLayer, double shapeReso)

// crossRoundDashed
public static GArea createCrossRoundDashed(double x, double y, double W
        , double L, double r, double shapeOutlineWidth, double dashLength,
        double dashGap, int CAP, int JOIN, double THETA, int gdsLayer,
        double shapeReso)

// crossRoundOutlineV2
public static GArea createCrossRoundOutlineV2(double x, double y,
        double W1, double L1, double W2, double L2, double r, double
        shapeOutlineWidth, double THETA, int gdsLayer, double shapeReso)

// crossRoundDashedV2
public static GArea createCrossRoundDashedV2(double x, double y, double
        W1, double L1, double W2, double L2, double r, double
        shapeOutlineWidth, double dashLength, double dashGap, int CAP, int
        JOIN, double THETA, int gdsLayer, double shapeReso)

// LshapeOutline
public static GArea createLshapeOutline(double x, double y, double W1,
        double L1, double W2, double L2, double shapeOutlineWidth, double
        THETA, int gdsLayer, double shapeReso)

// LshapeDashed
public static GArea createLshapeDashed(double x, double y, double W1,
        double L1, double W2, double L2, double shapeOutlineWidth, double
        dashLength, double dashGap, int CAP, int JOIN, double THETA, int
        gdsLayer, double shapeReso)

// LshapeRoundOutline

```

```

public static GArea createLshapeRoundOutline(double x, double y, double
    W1, double L1, double W2, double L2, double r, double
    shapeOutlineWidth, double THETA, int gdsLayer, double shapeReso)

// LshapeRoundDashed
public static GArea createLshapeRoundDashed(double x, double y, double
    W1, double L1, double W2, double L2, double r, double
    shapeOutlineWidth, double dashLength, double dashGap, int CAP, int
    JOIN, double THETA, int gdsLayer, double shapeReso)

// rectangleOutline
public static GArea createRectangleOutline(double xLL, double yLL,
    double xUR, double yUR, double shapeOutlineWidth, double THETA, int
    gdsLayer)

// rectangleDashed
public static GArea createRectangleDashed(double xLL, double yLL,
    double xUR, double yUR, double shapeOutlineWidth, double dashLength
    , double dashGap, int CAP, int JOIN, double THETA, int gdsLayer,
    double shapeReso)

// rectangleOutlineLH
public static GArea createRectangleOutlineLH(double xLL, double yLL,
    double L, double H, double shapeOutlineWidth, double THETA, int
    gdsLayer)

// rectangleDashedLH
public static GArea createRectangleDashedLH(double xLL, double yLL,
    double L, double H, double shapeOutlineWidth, double dashLength,
    double dashGap, int CAP, int JOIN, double THETA, int gdsLayer,
    double shapeReso)

// rectangleOutlineC
public static GArea createRectangleOutlineC(double xC, double yC,
    double L, double H, double shapeOutlineWidth, double THETA, int
    gdsLayer)

// rectangleDashedC
public static GArea createRectangleDashedC(double xC, double yC, double
    L, double H, double shapeOutlineWidth, double dashLength, double
    dashGap, int CAP, int JOIN, double THETA, int gdsLayer, double
    shapeReso)

// squareOutline
public static GArea createSquareOutline(double xLL, double yLL, double
    L, double shapeOutlineWidth, double THETA, int gdsLayer)

// squareDashed
public static GArea createSquareDashed(double xLL, double yLL, double L
    , double shapeOutlineWidth, double dashLength, double dashGap, int
    CAP, int JOIN, double THETA, int gdsLayer, double shapeReso)

// squareOutlineC
public static GArea createSquareOutlineC(double xC, double yC, double L
    , double shapeOutlineWidth, double THETA, int gdsLayer)

// squareDashedC

```

```

public static GArea createSquareDashedC(double xC, double yC, double L,
    double shapeOutlineWidth, double dashLength, double dashGap, int
    CAP, int JOIN, double THETA, int gdsLayer, double shapeReso)

// roundRectOutline
public static GArea createRoundRectOutline(double xLL, double yLL,
    double L, double H, double rX, double rY, double shapeOutlineWidth,
    double THETA, int gdsLayer, double shapeReso)

// roundRectDashed
public static GArea createRoundRectDashed(double xLL, double yLL,
    double L, double H, double rX, double rY, double shapeOutlineWidth,
    double dashLength, double dashGap, int CAP, int JOIN, double THETA
    , int gdsLayer, double shapeReso)

// roundRectOutlineC
public static GArea createRoundRectOutlineC(double xLL, double yLL,
    double L, double H, double rX, double rY, double shapeOutlineWidth,
    double THETA, int gdsLayer, double shapeReso)

// roundRectDashedC
public static GArea createRoundRectDashedC(double xLL, double yLL,
    double L, double H, double rX, double rY, double shapeOutlineWidth,
    double dashLength, double dashGap, int CAP, int JOIN, double THETA
    , int gdsLayer, double shapeReso)

// roundSquareOutline
public static GArea createRoundSquareOutline(double xLL, double yLL,
    double L, double r, double shapeOutlineWidth, double THETA, int
    gdsLayer, double shapeReso)

// roundSquareDashed
public static GArea createRoundSquareDashed(double xLL, double yLL,
    double L, double r, double shapeOutlineWidth, double dashLength,
    double dashGap, int CAP, int JOIN, double THETA, int gdsLayer,
    double shapeReso)

// roundSquareOutlineC
public static GArea createRoundSquareOutlineC(double xLL, double yLL,
    double L, double r, double shapeOutlineWidth, double THETA, int
    gdsLayer, double shapeReso)

// roundSquareDashedC
public static GArea createRoundSquareDashedC(double xLL, double yLL,
    double L, double r, double shapeOutlineWidth, double dashLength,
    double dashGap, int CAP, int JOIN, double THETA, int gdsLayer,
    double shapeReso)

// *****
//
// EXPORTING TO VARIOUS FILE FORMATS
//
// *****

//
// POINTSXY

```



```

//
// export POINTSXY
public static void exportPOINTSXY(Lib lib, String pointsXyFileName,
    boolean pointsXyColumn) throws IOException

// clear POINTSXY properties
public static void clearPOINTSXYproperties()

// set the POINTSXY output struct name
public static void setPOINTSXYstruct(String pointsXyStructName)

// sets the GDS layers to be converted to POINTSXY
public static void setPOINTSXYlayers(ArrayList<Integer> pointsXyLayers)

// GArea objects could have many closed shapes. Each closed shape
// contour, consisting of Point2D.Double values, is stored into an
// arrayList
public static ArrayList<ArrayList<Point2D.Double>> gAreaPointsXY(GArea
    g)

// Printing GArea objects: x-y coordinate pairs of shape vertices are
// printed. Each pair on a separate line. Blank line separates
// individual shapes.
public static void printGAreaPointsXY(GArea g)

// individual shapes: printing the Point2D coordinate values for a
// particular shape index in the arrayList
public static void printAlPt2D(ArrayList<ArrayList<Point2D.Double>>
    listOfShapeContours, int index)

// printing the Point2D coordinate values for a ALL shapes – each shape
// is separated by an extra blank line
public static void printAlPt2D(ArrayList<ArrayList<Point2D.Double>>
    listOfShapeContours)

```