

AARON HRYCIW

RAITH_GDSII MATLAB TOOLBOX USER GUIDE

VERSION 1.0

NATIONAL INSTITUTE FOR NANOTECHNOLOGY
EDMONTON, ALBERTA, CANADA



The Raith_GDSII MATLAB toolbox was developed by Aaron Hryciw at the National Institute for Nanotechnology (NINT), a joint initiative between the Government of Canada, the Government of Alberta, the National Research Council Canada (NRC), and the University of Alberta. This toolbox is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, you can obtain one at mozilla.org/MPL/2.0/.

PUBLISHED BY NATIONAL INSTITUTE FOR NANOTECHNOLOGY
EDMONTON, ALBERTA, CANADA

Typeset September 25, 2013

Contents

1	<i>Introduction</i>	9
2	<i>Quick-start guide</i>	13
3	<i>The Raith_element class</i>	17
4	<i>The Raith_structure class</i>	29
5	<i>The Raith_library class</i>	33
6	<i>The Raith_positionlist class</i>	45
7	<i>Extended techniques</i>	53
	<i>Index</i>	59

List of Figures

2.1	Cantilever 'polygon' element	13
2.2	'text' element to label cantilever length	13
2.3	Raith_structure object '10-um-cantilever'	13
2.4	Plotting the positionlist	14
2.5	Polygon defined in function <i>cantilever.m</i>	15
2.6	Positionlist plot of a cantilever array constructed using <i>cantilever.m</i>	16
2.7	3×3 “matrix copy” of the cantilever array	16
3.1	Example 'polygon' element	19
3.2	Example 'path' elements	19
3.3	Example 'dot' elements	20
3.4	Definition of angles used in 'arc' elements	21
3.5	Example 'arc' elements	21
3.6	Example 'circle' elements	22
3.7	Example 'ellipse' elements	22
3.8	Letter A rendered using the Raith default and Raith_element fonts	24
3.9	Example 'text' element	24
3.10	'sref' element transformations	25
3.11	Example 'sref' element	25
3.12	Example 'aref' element	26
3.13	Raith interpretation of AREF element with rotation	26
3.14	KLayout interpretation of AREF element with rotation	26
3.15	Raith dose factor colourmap	27
3.16	Text element plotted using the Raith_element.plot method	27
3.17	Text element plotted using the Raith_element.plotedges method	28
4.1	Example structure plotted using the Raith_structure.plot method.	31
4.2	Example structure plotted using the Raith_structure.plotedges method.	32
5.1	Raith_library.plot output with absent referenced structures	37
5.2	Raith_library.plot output with one absent referenced structure	37
5.3	Raith_library.plot output with no absent structures	37
5.4	Raith_library.plotedges with no absent referenced structures	38

6.1	Positionlist plotted using the Raith_positionlist.plot method	48
6.2	Positionlist plotted using the Raith_positionlist.plotedges method	49
6.3	Plot of positionlist working areas using Raith_positionlist.plotWA	49
6.4	Plot of positionlist working areas and writefields	50
6.5	Centring and matrix-copying positionlist entries with Raith_positionlist.centre	50

List of Tables

5.1	GDSII record types	41
5.2	GDSII data types	41

1

Introduction

THE RAITH_GDSII TOOLBOX provides a simple, versatile, and scriptable means of generating patterns for Raith electron-beam lithography (EBL) and focused ion beam (FIB) tools¹ using MATLAB².

Although relatively simple structures may be generated directly within the Raith software via the GDSII editing tools in the *Design* panel, this interface becomes cumbersome for complicated structures comprising many elements, or for arrays of structures with subtle variations in geometry. GDSII files can be scripted using third-party libraries³, but beam dose information and Raith curved elements (arcs, circles, and ellipses) are generally not supported⁴. Scripted generation of patterns using the ASCII-based .asc or .elm formats is possible⁵, but these files must be manually loaded into a GDSII hierarchy file in the Raith software; the GDSII file is backed up after *each* .asc/.elm file is added to the library, which can yield unacceptably long load-times if there are many structures to add.

To circumvent these issues, the Raith_GDSII toolbox may be used to generate both the GDSII hierarchy and positionlist files directly within MATLAB, with full support for Raith curved elements. The relevant objects may be manipulated using standard MATLAB functionality, making scripting easy. Furthermore, structures may be plotted using the standard Raith dose factor colourmap—from individual low-level elements to entire positionlists—to aid in visualisation and error-checking during the pattern design process.

1.1 The Raith_GDSII classes

THERE ARE FOUR CLASSES in the Raith_GDSII toolbox: Raith_element, Raith_structure, Raith_library, and Raith_positionlist. The first three classes reflect the structure of the GDSII stream format and are used to generate the GDSII library containing the structures referenced in the positionlist:

¹ www.raith.com

² www.mathworks.com/products/matlab/

³ E.g., pypi.python.org/pypi/python-gdsii

⁴ The libgds Python library at github.com/scholi/libgds does in fact encode the dose factor, but does not truly support Raith curved elements, instead implementing them as polygons or paths.

⁵ See §5.1.3 (Edit Menu) of the *Raith Software Reference Manual*, Version 5.0.

Raith_element

Used to define unnamed, low-level GDSII pattern elements. The following element types are supported:

polygon A closed, filled polygon. *'polygon'* elements are fractured into trapezoids by the Raith software before writing.

path A path of connected line segments. *'path'* elements may be either single-pixel lines or have a non-zero width.

dot A single-pixel dot, or series thereof.

arc A segment of a circular or elliptical path (Raith curved element). *'arc'* elements may be single-pixel lines, have a non-zero width, or be filled (i.e., a circular or elliptical segment).

circle A circle or disk (Raith curved element). *'circle'* elements may be single-pixel lines, have a non-zero width, or be filled (i.e., a disk).

ellipse An ellipse or elliptical disk (Raith curved element).

'ellipse' elements may be single-pixel lines, have a non-zero width, or be filled (i.e., an elliptical disk).

text A line of text rendered as simply-connected polygons⁶.

sref A structure reference. *'sref'* elements refer to named Raith_structure objects, and may optionally apply transformations (magnification, rotation, reflection across the *u* axis)⁷.

aref An array reference. *'aref'* elements generate a rectangular array of named Raith_structure objects, and may optionally apply transformations (magnification, rotation, reflection across the *u* axis)^{7,8}.

Elements of type *'arc'*, *'circle'*, and *'ellipse'* are implemented as Raith curved elements, with a curved beam path which in general consists of concentric ellipses, rather than being fractured into trapezoids.

Raith_structure

Used to define named structures, comprising collections of Raith_element objects.

Raith_library

Used to define a GDSII library, comprising a collection of uniquely named Raith_structure objects, and to write a Raith-readable GDSII hierarchy (.csf) file.

Raith_positionlist

Used to define a positionlist, comprising chip-level references to Raith_structure objects in a Raith_library, and to write a Raith-readable positionlist (.pls) file.

⁶ Using simply-connected polygons for text shapes prevents the interiors of letters (e.g., A,B,D) from being released if there is a subsequent undercut step.

⁷ The little-used *absolute magnification* and *absolute rotation* transformations in the GDSII specification are not supported by the Raith_GDSII toolbox.

⁸ The Raith software's interpretation of *'aref'* objects differs somewhat from the GDSII specification. See §3.2.9.

1.2 *Software use and bug reporting*

USE OF THE Raith_GDSII TOOLBOX is subject to the terms of the Mozilla Public License, v. 2.0⁹.

⁹ mozilla.org/MPL/2.0/

The latest version of the Raith_GDSII toolbox may be downloaded from the National Research Council Canada GitHub repository¹⁰.

¹⁰ github.com/nrc-cnrc/Raith_GDSII

Please send comments, bug reports, and future update suggestions to Aaron Hryciw at aaron.hryciw@nrc-cnrc.gc.ca.

1.3 *Citing Raith_GDSII*

Please cite the Raith_GDSII MATLAB toolbox in any publication for which you found it useful by including the text “The Raith_GDSII MATLAB toolbox was developed at the National Institute for Nanotechnology; it is available at github.com/nrc-cnrc.” in a footnote or endnote, as appropriate.

1.4 *Installation*

TO INSTALL the Raith_GDSII toolbox, simply place the four Raith_GDSII class definitions (*Raith_element.m*, *Raith_structure.m*, *Raith_library.m*, and *Raith_positionlist.m*) in a folder on your MATLAB path. A full description of these classes is contained in §§3–6. To get started, however, the following section outlines a typical (albeit brief) workflow.

2

Quick-start guide

AS A SIMPLE EXAMPLE, let us construct a pattern of a cantilever for use with a positive-tone resist (e.g., ZEP, PMMA). First, define a polygon element for a 10- μm -long, 1- μm -wide cantilever with 3 μm spacing between the cantilever and the edge of the window:

```
% obj=Raith_element('polygon',layer,uv,DF)
% uv is a 2xn matrix of polygon vertices [u_values;v_values]
E=Raith_element('polygon',0,[0 13 13 0 0 10 10 0 0; ...
                             0 0 7 7 4 4 3 3 0],1.3);

axis equal;
E.plot;
```

Next, create a text element to label the cantilever length. Since both of these elements will belong to the same Raith_structure object, we can make E an array of Raith_element objects:

```
% obj=Raith_element('text',layer,uv_0,h,angle,uv_align, ...
%   textlabel,DF)
E(2)=Raith_element('text',0,[14 3.5],3,0,[0 1],'10',1.5);
clf;
axis equal;
E(2).plot;
```

We now bundle these elements into a named structure:

```
% obj=Raith_structure(name,elements)
S=Raith_structure('10-um-cantilever',E);
clf;
axis equal;
S.plot;
```

This structure can now be used in a Raith_library object, used to create the GDSII hierarchy (.csf file) which will be loaded into the Raith software:

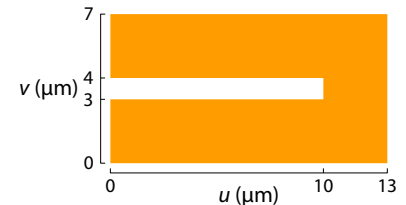


Figure 2.1: Cantilever 'polygon' element

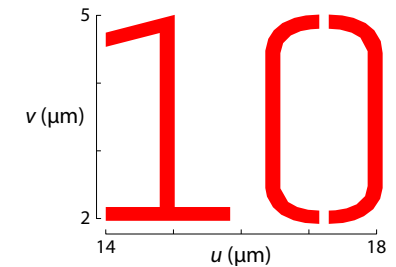


Figure 2.2: 'text' element to label cantilever length

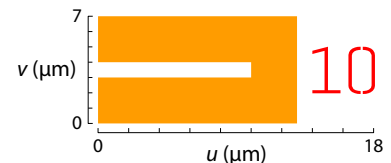


Figure 2.3: Raith_structure object '10-um-cantilever'

```
% obj=Raith_library(name,structures)
L=Raith_library('cantilevers',S);
L.writegds;

Checking for missing structures...OK.
Writing C:\Users\Public\Documents\cantilevers.csf...
Header information
Structure 1: 10-um-cantilever
GDSII library cantilevers.csf successfully written.
```

Next, we create a positionlist using a `Raith_positionlist` object. We specify a $100\ \mu\text{m} \times 100\ \mu\text{m}$ writefield and a $10\ \text{mm} \times 10\ \text{mm}$ chip, and assume that the path of *cantilevers.csf* will be *F:\Raith* on the Raith computer:

```
% obj=Raith_positionlist(library,csf_path,WF,chipUV)
P=Raith_positionlist(L,'F:\Raith\cantilevers.csf',[100 100], ...
[10 10]);
% Append a structure to the positionlist using
% P.append(structname,uv_c,DF,WA,[layers]). WA defines the
% working area, WA=[u_min v_min u_max v_max], in um. Argument
% layers is optional, and defaults to exposing all layers
% present in structure.
P.append('10-um-cantilever',[5 5],1,[-50 -50 50 50]);
clf;
P.plot; % Plot structures and chip boundaries
P.plotWF; % Plot writefield as green, dotted line
P.writepls; % Write cantilevers.pls to current directory
```

To use these files in an EBL or FIB session, place *cantilevers.csf* in *F:\Raith* on the Raith computer, open *cantilevers.csf* via **Design panel**→**File**→**Open...** in the Raith software, and open *cantilevers.pls* via **File**→**Open positionlist**. After the usual preliminary steps (origin and angle correction, aperture alignment, stigmation, focusing, beam current measurement, etc.), the positionlist may be scanned as normal.

THE ABOVE EXAMPLE illustrates the main functionality of the Raith_GDSII toolbox. In practice, however, structure definitions could be parametrised to facilitate script-based generation of many devices with similar, though distinct, geometries. For example, we could create a function (*cantilever.m*) which takes the cantilever length, cantilever width, window width (in μm) as arguments and returns a `Raith_structure` object:

```
function S=cantilever(L_c,w_c,w_w)
%
% function S=cantilever(L_c,w_c,w_w)
%
```

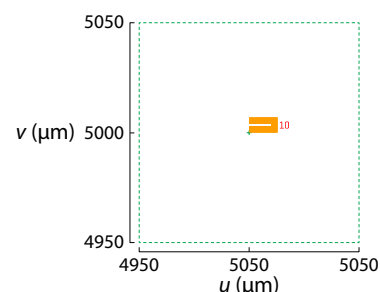


Figure 2.4: Plotting the positionlist. The writefield boundary is marked by a green dotted line, with the centre marked with a +. Axis limits were chosen to show the structure at chip centre.

```

% Create a cantilever pattern in layer 0 with unit DF.
% A label indicating cantilever length is included,
% with DF = 1.5.
%
% Arguments:
%
% L_c - cantilever length (um)
% w_c - cantilever width (um)
% w_w - window width (um)
%
% Return value:
%
% S - Raith_structure object containing labelled
%     cantilever
%

% Define vertices of cantilever polygon
u1=L_c+w_w;
u2=L_c;

v1=2*w_w+w_c;
v2=w_w+w_c;
v3=w_w;

u=[0 u1 u1 0 0 u2 u2 0 0];
v=[0 0 v1 v1 v2 v2 v3 v3 0];

E=Raith_element('polygon',0,[u;v],1);

% Define text label for cantilever length
% Text height is hard-coded at 3 um
% Label placed to left of cantilever
E(2)=Raith_element('text',0,[-2 v1/2],3,0,[2 ...
    1],num2str(L_c),1.5);

name=[num2str(L_c) '-um-cantilever'];

S=Raith_structure(name,E);

end

```

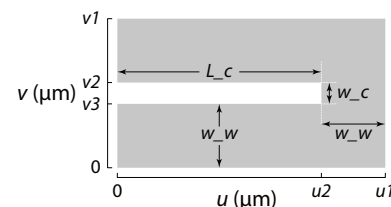


Figure 2.5: Polygon defined in function *cantilever.m*

Using this function, it is simple to generate an array of labelled cantilevers with varying lengths while keeping the window widths constant:

```

L_c=4:2:30; % Cantilever lengths, from 4 um to 30 um in 2 ...
            um steps
w_c=1; % Cantilever width (um)
w_w=3; % Window width (um)

% Loop to construct all cantilever structures
for k=1:length(L_c)
    S(k)=cantilever(L_c(k),w_c,w_w);
end

L2=Raith_library('cantilevers',S);

P2=Raith_positionlist(L2,'F:\Raith\cantilevers.csf', ...

```

```
[100 100],[5 5]); % Positionlist object for a 5 x 5 mm chip

Dv=0.010; % Vertical centre-to-centre distance (mm)

for k=1:length(L_c)
    P2.append(S(k).name,[2.5 2.5+k*Dv],1,[-50 -50 50 50]);
end

P2.plot; % Plot structures and chip boundaries
```

As a final example of a useful Raith_GDSII toolbox feature, note that Raith_positionlist objects have a **centre** method which shifts an entire positionlist to centre it on the chip. This method also takes an optional argument to create a “matrix copy” array (to use Raith software terminology) of the positionlist, with the entire matrix centred on the chip; this is useful, for example, to create multiple copies of the pattern on the chip for subsequent cleaving of the specimen into sub-chips.

Given the above positionlist P2, we can create a 3×3 array of this pattern on the 5×5 mm chip via:

```
% "Matrix copy" current positionlist into a 3 x 3 array
% centred on the 5 x 5 mm chip.
P2.centre([3 3]);

P2.plot; % Plot updated positionlist
```

THE PURPOSE OF THIS SECTION has been to demonstrate the major features of the Raith_GDSII toolbox, illustrating how to create, preview, and edit patterns for Raith electron- and ion-beam lithography tools—all within MATLAB. By enabling users to generate all files necessary for a beamwriting session within the widely available (and programmable) MATLAB environment, the Raith_GDSII toolbox helps to shorten and simplify design cycle iterations, especially for complicated patterns. The remainder of this document is devoted to a thorough explanation of the Raith_GDSII toolbox classes.



Figure 2.6: Positionlist plot of a cantilever array constructed using *cantilever.m*

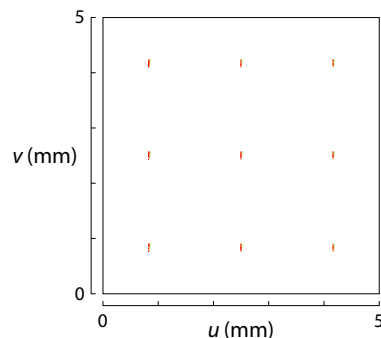


Figure 2.7: Positionlist plot of the pattern in Figure 2.6 “matrix copied” into a 3×3 array using the Raith_positionlist.centre method

3

The Raith_element class

Class overview: Raith_element

Properties (public)

type String specifying type of element

data Structure with fields defining element geometry

Methods

plot Plot element as filled polygons

plotedges Plot element as unfilled polygons

RAITH_ELEMENT OBJECTS define low-level, unnamed patterns, collections of which are bundled together to form named structures in the GDSII library.

3.1 *Properties*

type: String specifying type of element; allowed values are 'polygon', 'path', 'dot', 'arc', 'circle', 'ellipse', 'text', 'sref', or 'aref'

data: Struct array containing additional record data for element; allowed field names and typing of values are determined by the element **type** (see §3.2)

3.2 *Constructors*

```
E=Raith_element('polygon', layer, uv, DF)
E=Raith_element('path', layer, uv, w, DF)
E=Raith_element('dot', layer, uv, DF)
E=Raith_element('arc', layer, uv_c, r, theta, angle, w, N, DF)
E=Raith_element('circle', layer, uv_c, r, w, N, DF)
E=Raith_element('ellipse', layer, uv_c, r, w, angle, N, DF)
E=Raith_element('text', layer, uv_0, h, angle, uv_align, textlabel, DF)
E=Raith_element('sref', name, uv_0, [mag, angle, reflect])
E=Raith_element('aref', name, uv_0, n_colrow, a_colrow, [mag, angle, reflect])
```

THE ABOVE CONSTRUCTORS may be used to create `Raith_element` objects. The first argument sets the element **type** property, followed by a list of arguments comprising the fields of the **data** property (a MATLAB structure), which vary depend on the **type**. Arguments shown in brackets are optional.

Alternately, an empty, argumentless `Raith_element` object may be called, with the **type** and **data** properties assigned afterward. For example:

```
E=Raith_element;
E.type='polygon';
E.data.layer=0;
E.data.uv=[0 1 1 0 0;0 0 1 1 0];
E.data.DF=1.5;
```

The above is equivalent to

```
E=Raith_element('polygon',0,[0 1 1 0 0;0 0 1 1 0],1.5);
```

BY DEFAULT, all properties are checked for correctness (typing, allowed values, size) before being assigned, whether the `Raith_element` object is created with a constructor or its properties are amended individually. Descriptions of the nine `Raith_element` types are given in the following subsections.

3.2.1 Polygon element

Description

Closed, filled polygon

Constructor

```
E=Raith_element('polygon',layer,uv,DF)
```

Properties

- type:** 'polygon' (string)
- data.layer:** GDSII layer; allowed values are 0–63
- data.uv:** $2 \times n$ matrix $[u;v]$ of polygon vertices (μm)
- data.DF:** Dose factor for polygon

Note

If the first and last vertices in **data.uv** are not the same (i.e., an open polygon), **data.uv** is amended to close the polygon and a warning is issued.

Example

```
E=Raith_element('polygon',0,[0 2 2 1 1 0 0; ...
    0 0 1 1 2 2 0],1.3);
```

3.2.2 Path element**Description**

Path of line segments

Constructor

```
E=Raith_element('path',layer,uv,w,DF)
```

Properties

- type:** 'path' (string)
- data.layer:** GDSII layer; allowed values are 0–63
- data.uv:** $2 \times n$ matrix $[u;v]$ of path vertices (μm)
- data.w:** Width of path (μm); a value of zero yields single-pixel line; a negative value is considered to be the same as zero by the Raith software (single-pixel line)
- data.DF:** Dose factor for path

Note

The interpretation of a negative value for GDSII path WIDTH records differs between the Raith software and the standard GDSII specification. In the former, a negative width is considered the same as zero width (single-pixel line); in the latter, a negative value denotes an *absolute* width, that is, a fixed width which is not affected by magnification of any parent structure ('sref' or 'aref' elements).

Example

```
E1=Raith_element('path',0,[0 0 1 1 2;1 0 0 1 1],0,1.3);
E2=Raith_element('path',0,[0 0 1 1 2;1 0 0 1 1],0.2,1.3);
```

3.2.3 Dot element**Description**

Single-pixel dot(s)

Constructor

```
E=Raith_element('dot',layer,uv,DF)
```

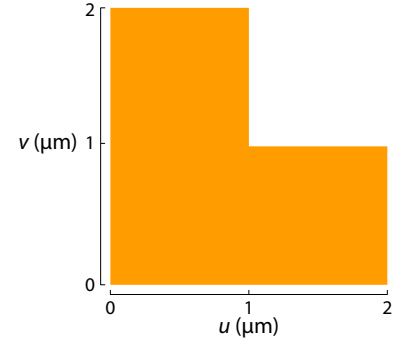


Figure 3.1: Example 'polygon' element

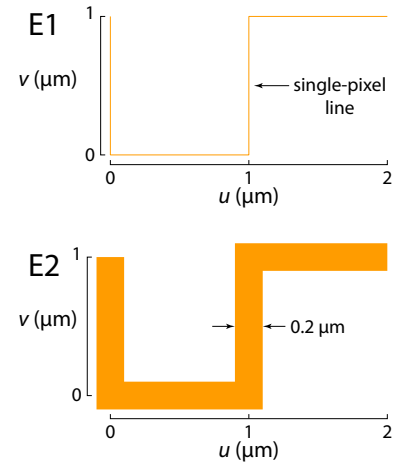


Figure 3.2: Example 'path' elements
Element E1: data.w = 0
Element E2: data.w = 0.2

Properties

- type:** 'dot' (string)
- data.layer:** GDSII layer; allowed values are 0–63
- data.uv:** $2 \times n$ matrix $[u;v]$ of dot positions (μm)
- data.DF:** Dose factor(s) for dot(s); if scalar, all dots given in **data.uv** have the same dose factor; if vector, **data.DF** must be the same length as **data.uv**, and specifies the dose factor of each dot

Example

```
E1=Raith_element('dot',0,[0 2 2 0;0 0 1 1],1.3);
E2=Raith_element('dot',0,[0 2 2 0;0 0 1 1],[0 0.5 1.0 1.5]);
```

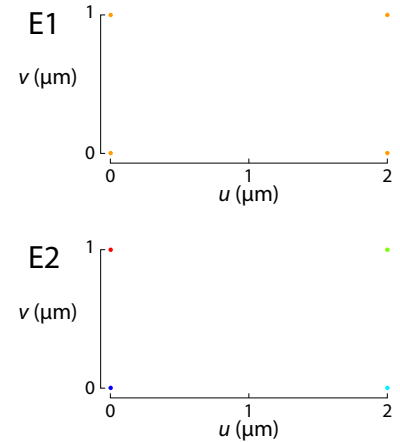


Figure 3.3: Example 'dot' elements
 Element E1: scalar data.DF
 Element E2: vector data.DF

3.2.4 Arc element

Description

Segment of circular or elliptical path (Raith curved element)

Constructor

```
E=Raith_element('arc',layer,uv_c,r,theta,angle,w,N,DF)
```

Properties

- type:** 'arc' (string)
- data.layer:** GDSII layer; allowed values are 0–63
- data.uv_c:** Arc centre; 1×2 vector $[u_c \ v_c]$ (μm)
- data.r:** Radius of arc; may be scalar for a circular arc, or a 1×2 vector denoting semi-axes, $[a \ b]$, for an elliptical arc (μm)
- data.theta:** Starting and ending angles of arc with respect to axis defined by **data.angle** argument, counter-clockwise positive; 1×2 vector $[\theta_1 \ \theta_2]$ (degrees)
- data.angle:** Angle of rotation ϕ between positive u -axis and $\theta = 0$ axis (degrees)
- data.w:** Arc linewidth (μm); if empty, arc is a filled elliptical disk segment; if zero, arc is a single-pixel line; if non-zero, arc has a width; a negative value is considered to be the same as empty by the Raith software (filled elliptical disk segment)
- data.N:** Number of vertices along arc length

data.DF: Dose factor for arc

Note

Arc elements are interpreted by the Raith software using the following parametric equations:

$$u(\theta) = u_c + a \cos(\theta) \cos(\phi) - b \sin(\theta) \sin(\phi) \quad (3.1)$$

$$v(\theta) = v_c + a \cos(\theta) \sin(\phi) + b \sin(\theta) \cos(\phi) \quad (3.2)$$

with $\theta \in [\theta_1, \theta_2]$, spaced linearly over **data.N** points. As such, for elliptical arcs (i.e., $a \neq b$), θ is a *parametric* angle, and does not in general correspond to the angle from the positive u axis (assuming $\phi = 0$). To convert between the polar angle from the ellipse centre ϕ' and the parametric angle θ required by **data.theta**, use

$$\tan \theta = \frac{a}{b} \tan \phi' \quad (3.3)$$

Note that $\theta = \phi'$ for multiples of 90° .¹

Example

```
E1=Raith_element('arc',0,[0 3],[2 1],[0 120],10,[],7,1.3);
E2=Raith_element('arc',0,[0 1.5],[2 1],[0 120],10,0,7,1.3);
E3=Raith_element('arc',0,[0 0],[2 1],[0 120],10,0.2,7,1.3);
```

3.2.5 Circle element

Description

Circle or circular disk (Raith curved element)

Constructor

```
E=Raith_element('circle',layer,uv_c,r,w,N,DF)
```

Properties

type: 'circle' (string)

data.layer: GDSII layer; allowed values are 0–63

data.uv_c: Circle centre; 1×2 vector $[u_c \ v_c]$ (μm)

data.r: Radius of circle (μm)

data.w: Circle linewidth (μm); if empty, circle is filled (disk); if zero, circle is a single-pixel line; if non-zero, circle has a width; a negative value is considered to be the same as empty by the Raith software (disk)

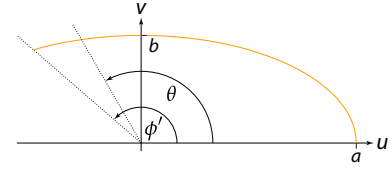


Figure 3.4: Angles used in 'arc' elements. For $a=2$ and $b=1$, $\theta = 120^\circ$ corresponds to $\phi' = 139.1^\circ$.

¹ en.wikipedia.org/wiki/Ellipse

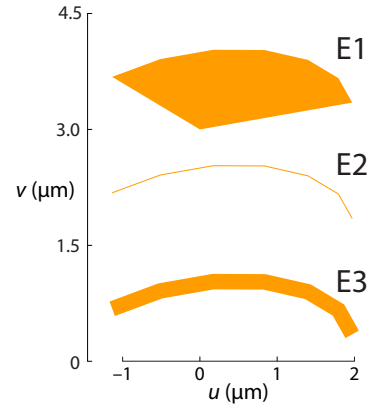


Figure 3.5: Example 'arc' elements

Element E1: data.w = []

Element E2: data.w = 0

Element E3: data.w = 0.2

data.N: Number of vertices along circle circumference

data.DF: Dose factor for circle

Example

```
E1=Raith_element('circle',0,[0 0],1,[],60,1.3);
E2=Raith_element('circle',0,[3 0],1,0,60,1.3);
E3=Raith_element('circle',0,[6 0],1,0.2,60,1.3);
```

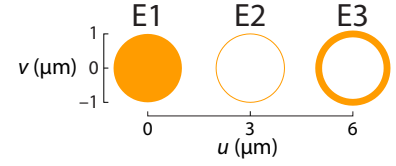


Figure 3.6: Example 'circle' elements

Element E1: data.w = []

Element E2: data.w = 0

Element E3: data.w = 0.2

3.2.6 Ellipse element

Description

Ellipse or elliptical disk (Raith curved element)

Constructor

```
E=Raith_element('ellipse',layer,uv_c,r,w,angle,N,DF)
```

Properties

type: 'ellipse' (string)

data.layer: GDSII layer; allowed values are 0–63

data.uv_c: Ellipse centre; 1×2 vector $[u_c \ v_c]$ (μm)

data.r: Semi-axes of ellipse; 1×2 vector $[a \ b]$ (μm); a corresponds to the semi-axis in the **data.angle** direction

data.w: Ellipse linewidth (μm); if empty, ellipse is filled (elliptical disk); if zero, ellipse is a single-pixel line; if non-zero, ellipse has a width; a negative value is considered to be the same as empty by the Raith software (elliptical disk)

data.angle: Angle of rotation ϕ between positive u -axis and a semi-axis (degrees)

data.N: Number of vertices along ellipse circumference

data.DF: Dose factor for ellipse

Example

```
E1=Raith_element('ellipse',0,[0 6],[2 1],[],10,60,1.3);
E2=Raith_element('ellipse',0,[0 3],[2 1],0,10,60,1.3);
E3=Raith_element('ellipse',0,[0 0],[2 1],0.2,10,60,1.3);
```

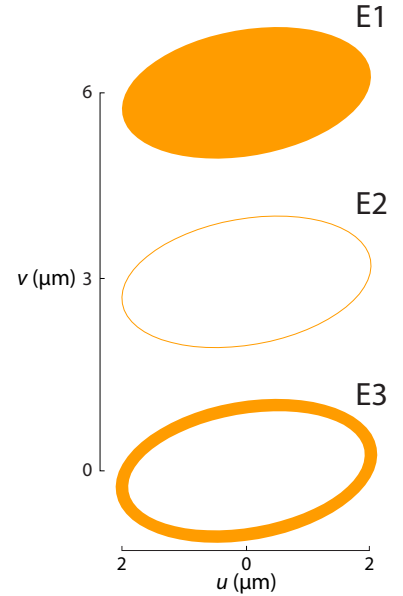


Figure 3.7: Example 'ellipse' elements

Element E1: data.w = []

Element E2: data.w = 0

Element E3: data.w = 0.2

3.2.7 Text element

Description

Text rendered as simply-connected polygons

Constructor

```
E=Raith_element('text', layer, uv_0, h, angle, uv_align, textlabel, DF)
```

Properties

type: 'text' (string)

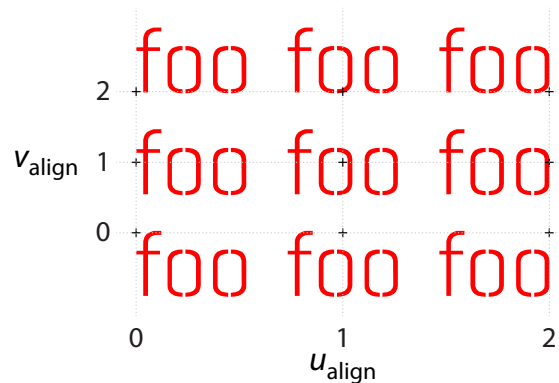
data.layer: GDSII layer; allowed values are 0–63

data.uv_0: Text anchor point [u_0 v_0] (μm)

data.h: Height of capital letters (μm)

data.angle: Angle of rotation of text with respect to positive u -axis (degrees)

data.uv_align: Alignment of text with respect to anchor point; 1×2 vector [u_{align} v_{align}]; allowed values are 0 (left/top), 1 (centre), and 2 (right/bottom), as follows (the + symbols denote the text anchor points):



data.textlabel: Text to be written (string). The allowed characters, shown as rendered, are:

```
` 1 2 3 4 5 6 7 8 9
0 - = q w e r t y u
i o p [ ] \ a s d f
g h j k l ; ' z x c
v b n m , . / ~ ! @
# $ % ^ & * [ ] _ +
Q W E R T Y U I O P
{ } | A S D F G H J
K L : " Z X C V B N
M < > ?
```

in addition to the space character (). When rendered, text is kerned using a look-up table (text is not fixed width).

data.DF: Dose factor for text

Note

A simply-connected font used in Raith_element 'text' elements to avoid the problem of symbol segments being released during a sacrificial layer etch. As an example, considering etching the letter *A* through the device layer of a silicon-on-insulator chip. In the default Raith font, the triangular centre of the letter *A* is not connected to the surrounding plane. If the underlying buried oxide layer was subsequently etched away isotropically for sufficiently long (e.g., in buffered-oxide etch), the central triangle would be released, potentially landing on a critical feature of the chip. A letter *A* rendered as a Raith_element 'text' element does not encounter this problem due to its simply-connected nature. The Raith_element 'text' element font is based on the Geogrotesque² and Geogrotesque Stencil³ fonts.

Example

```
E=Raith_element('text',0,[0 0],1,30,[1 1],'Raith_GDSII',1.3);
```



Figure 3.8: Comparison between letter *A* rendered using the Raith default font (left) and Raith_element font (right)

² www.emtype.net/geogrotesque_01.php

³ www.emtype.net/geogrotesque_Stencil_01.php

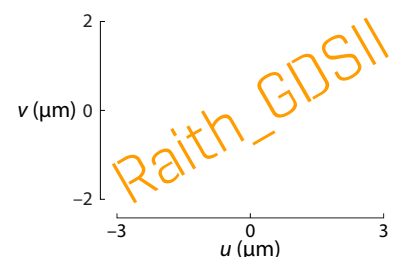


Figure 3.9: Example 'text' element

3.2.8 Structure reference element

Description

Reference to a named structure, with optional transformations

Constructor

```
E=Raiith_element('sref', name, uv_0, [mag, angle, reflect])
```

Properties

- type:** 'sref' (string)
- data.name:** Name of structure being referenced (string)
- data.uv_0:** Structure origin; 1×2 vector $[u_0 \ v_0]$ (μm)
- data.mag:** Magnification factor [optional]; default is no magnification (**data.mag** = 1)
- data.angle:** Angle of rotation, counter-clockwise positive (degrees) [optional]; default is no rotation (**data.angle** = 0)
- data.reflect:** Boolean flag (0 or 1) for reflecting about u -axis before other transformations [optional]; default is no reflection (**data.reflect** = 0)

Note

Transformations are applied in the following order: 1. scaling, mirroring; 2. rotation; 3. insertion. When 'sref' elements are plotted using the **Raiith_element.plot** method, the origin is marked with a + sign, labelled with **data.name**: since the structure being referenced is not part of the Raiith_element 'sref' object itself, the full hierarchy cannot be plotted. To view the full hierarchy, the structure must be plotted using the **Raiith_library.plot** method.

Example

```
E=Raiith_element('sref', 'foo', [10 20], 2, 30);
```

3.2.9 Array reference element

Description

Rectangular array of named structures, with optional transformations

Constructor

```
E=Raiith_element('aref', name, uv_0, n_colrow, a_colrow, [mag, angle, reflect])
```

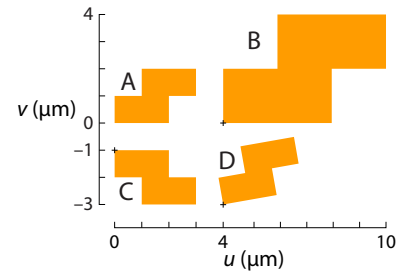


Figure 3.10: 'sref' element transformations. **data.uv_0** values for the transformed structures are marked with + signs.

- A: Structure being referenced
- B: **data.mag** = 2
- B: **data.reflect** = 1
- D: **data.angle** = 10

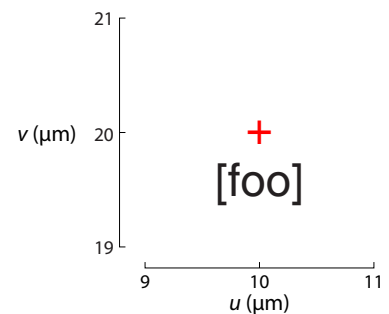


Figure 3.11: Example 'sref' element, as plotted using the **Raiith_element.plot** method

Properties

- type:** 'aref' (string)
- data.name:** Name of structure being referenced (string)
- data.uv_0:** Structure origin; 1×2 vector $[u_0 \ v_0]$ (μm)
- data.n_colrow:** Number of columns and rows in array; 1×2 vector $[n_{\text{columns}} \ n_{\text{rows}}]$
- data.a_colrow:** Spacing of columns and rows; 1×2 vector $[a_{\text{columns}} \ a_{\text{rows}}]$ (μm)
- data.mag:** Magnification factor [optional]; default is no magnification (**data.mag** = 1)
- data.angle:** Angle of rotation, counter-clockwise positive (degrees) [optional]; default is no rotation (**data.angle** = 0)
- data.reflect:** Boolean flag (0 or 1) for reflecting about u -axis before other transformations [optional]; default is no reflection (**data.reflect** = 0)

Note

Transformations are applied in the following order: 1. scaling, mirroring; 2. rotation; 3. insertion. When 'aref' elements are plotted using the **Raith_element.plot** method, the origins of the instances are marked with + signs, labelled with **data.name**: since the structure being referenced is not part of the Raith_element 'aref' object itself, the full hierarchy cannot be plotted. To view the full hierarchy, the structure must be plotted using the **Raith_library.plot** method.

Example

```
E=Raith_element('aref','foo',[10 20],[4 3],[3 2],[],30);
```

It is important to note that the Raith software interprets GDSII AREF elements differently than the GDSII specification suggests. In particular, the Raith software applies rotation operations *both* to the structures being referenced and the lattice vectors defining the rectangular array. In contrast, the GDSII specification applies the rotation only to the structures; the lattice of origins for the referenced structures are fully specified using the number of rows and columns in addition to three anchor points which are calculated *after* all transformations have been applied. This variation in interpretation can result in identical AREF elements appearing differently when viewed using the Raith software versus other GDSII editors, such as KLayout; Figures 3.13 and 3.14 illustrate this behaviour.

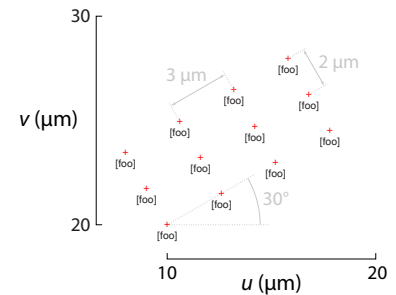


Figure 3.12: Example 'aref' element, as plotted using the **Raith_element.plot** method

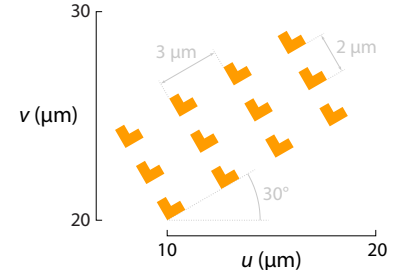


Figure 3.13: Raith interpretation of the AREF element in Figure 3.12, for an L-shaped structure named 'foo'

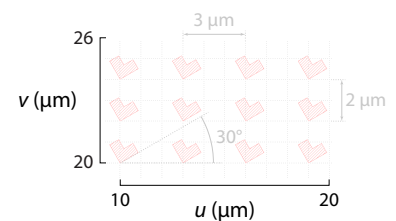


Figure 3.14: KLayout interpretation of the AREF element in Figure 3.12, for an L-shaped structure named 'foo'

3.3 Methods

3.3.1 *Raith_element.plot* method

Description

Plot `Raith_element` object with Raith dose factor colouring. Elements are displayed as filled polygons, where applicable ('`polygon`'; '`path`' with non-zero `data.w`; '`arc`', '`circle`', and '`ellipse`' with empty `data.w`; '`text`').

Syntax

```
plot
plot (M)
plot (M, scDF)
```

Arguments

- M Augmented transformation matrix to be applied to element [optional]; see §§5.4.1–5.4.4
- scDF Overall multiplicative scaling factor for dose factor specified in `data.DF` [optional]

Note

Normally, `Raith_element.plot` is called without arguments, to display the `Raith_element` object as it would appear in the Raith software. The optional arguments `M` and `scDF` are used internally, when `Raith_element.plot` is called by `Raith_structure.plot`, `Raith_library.plot`, or `Raith_positionlist.plot`.

Calling `Raith_element.plot` does not change the current axis scaling; issue an `axis equal` command to ensure that the element is displayed in the figure correctly.

Example

```
E=Raith_element('text',0,[0 0],1,0,[0 2],'B',1.3);
E.plot;
axis equal;
```

3.3.2 *Raith_element.plotedges* method

Description

Plot `Raith_element` object outlines with Raith dose factor colouring. Elements are displayed as unfilled polygons, where applicable ('`polygon`'; '`path`' with non-zero `data.w`; '`arc`', '`circle`', and '`ellipse`' with empty `data.w`; '`text`').

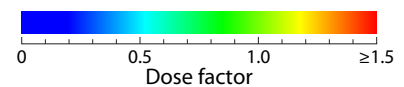


Figure 3.15: Raith dose factor colourmap

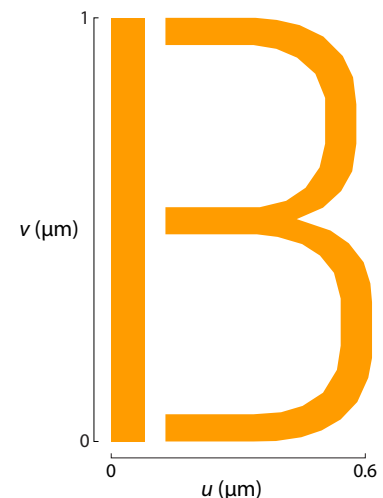


Figure 3.16: Text element plotted using the `Raith_element.plot` method

Syntax

```

plotedges
plotedges(M)
plotedges(M,scDF)

```

Arguments

- M Augmented transformation matrix to be applied to element [optional]; see §§5.4.1–5.4.4
- scDF Overall multiplicative scaling factor for dose factor specified in **data.DF** [optional]

Note

Normally, **Raith_element.plotedges** is called without arguments. The optional arguments M and scDF are used internally, when **Raith_element.plotedges** is called by **Raith_structure.plotedges**, **Raith_library.plotedges**, or **Raith_positionlist.plotedges**.

Calling **Raith_element.plotedges** does not change the current axis scaling; issue an `axis equal` command to ensure that the element is displayed in the figure correctly.

Example

```

E=Raith_element('text',0,[0 0],1,0,[0 2],'B',1.3);
E.plotedges;
axis equal;

```

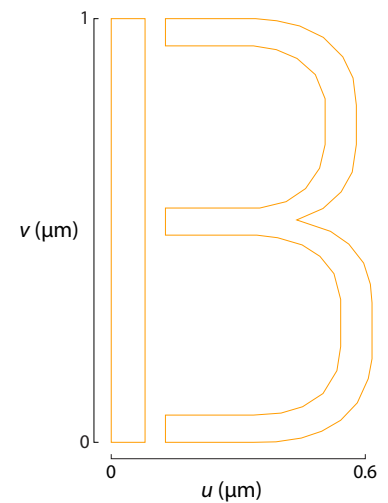


Figure 3.17: Text element plotted using the **Raith_element.plotedges** method

4

The Raith_structure class

Class overview: Raith_structure

Properties (public)

name String specifying name of structure

elements Array of Raith_element objects in structure

Properties (private set access)

reflist Cell array of referenced structure names

Methods

plot Plot structure as filled polygons

plotedges Plot structure as unfilled polygons

RAITH_STRUCTURE OBJECTS define named structures composed of low-level elements (Raith_element objects). Structures are packaged together in a GDSII hierarchy (library), and are the objects referred to in positionlist entries.

4.1 Properties

4.1.1 Public properties

name: String specifying name of structure. Maximum length is 127 characters. Allowed characters are A–Z, a–z, 0–9, underscore (_), period (.), dollar sign (\$), question mark (?), and hyphen (-).¹

elements: Array of Raith_element objects in structure. Raith_element arrays are created using standard MATLAB notation.

¹ The Raith software is somewhat more relaxed as regards structure names than the GDSII specification, which does not allow periods or hyphens and has a maximum length of 32.

4.1.2 Private set-access properties

reflist: Cell array of structure names referenced by 'sref' or 'aref' elements within the structure. **reflist** is automatically updated whenever **elements** is amended.

4.2 Constructor

```
S=Raith_structure(name,elements)
```

In a GDSII library, structures are defined by giving an arbitrary collection of elements a name; this simple conceptual framework is followed by the Raith_structure object constructor. By default, all properties are checked for correctness (typing, allowed values, size) before being assigned, whether the Raith_structure object is created with a constructor or its properties are amended individually.

Arguments

- name** String specifying name of structure. Maximum length is 127 characters. Allowed characters are A–Z, a–z, 0–9, underscore (_), period (.), dollar sign (\$), question mark (?), and hyphen (-). Illegal characters are replaced with underscores (with a warning issued).
- elements** Array of Raith_element objects in structure. Raith_element arrays are created using standard MATLAB notation (see “Example” below).

Example

```
% Optical racetrack resonator
E(1)=Raith_element('arc',0,[2 0],3,[-90 90],0,0.3,200,1.3);
E(2)=Raith_element('arc',0,[-2 0],3,[90 270],0,0.3,200,1.3);
E(3)=Raith_element('path',0,[-2 2;3 3],0.3,1.3);
E(4)=Raith_element('path',0,[-2 2;-3 -3],0.3,1.3);
S=Raith_structure('racetrack',E);
```

4.3 Methods

4.3.1 Raith_structure.plot method

Description

Plot Raith_structure object with Raith dose factor colouring. Elements are displayed as filled polygons, where applicable ('polygon'; 'path' with non-zero **data.w**; 'arc', 'circle', and 'ellipse' with empty **data.w**; 'text'). All elements in the structure are plotted, regardless of **data.layer** value.

Syntax

```
plot
plot (M)
plot (M, scDF)
```

Arguments

- M Augmented transformation matrix to be applied to structure [optional]; see §§5.4.1–5.4.4
- scDF Overall multiplicative scaling factor applied to dose factors of all elements in structure [optional]

Note

Normally, **Raith_structure.plot** is called without arguments, to display the Raith_structure object as it would appear in the Raith software. The optional arguments M and scDF are used internally, when **Raith_structure.plot** is called by **Raith_library.plot** or **Raith_positionlist.plot**.

Calling **Raith_structure.plot** does not change the current axis scaling; issue an `axis equal` command to ensure that the element is displayed in the figure correctly.

Example

Given the Raith_structure object s defined in the above “Constructor” section:

```
s.plot;
axis equal;
```

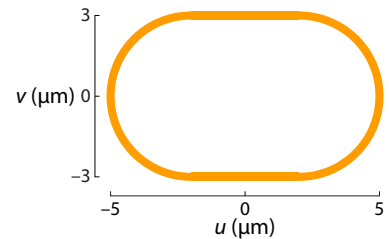


Figure 4.1: Racetrack resonator structure plotted using the **Raith_structure.plot** method

4.3.2 Raith_structure.plotedges method

Description

Plot Raith_structure object outlines with Raith dose factor colouring. Elements are displayed as unfilled polygons, where applicable ('polygon'; 'path' with non-zero **data.w**; 'arc', 'circle', and 'ellipse' with empty **data.w**; 'text'). All elements in the structure are plotted, regardless of **data.layer** value.

Syntax

```
plotedges
plotedges (M)
plotedges (M, scDF)
```

Arguments

- M Augmented transformation matrix to be applied to structure [optional]; see §§5.4.1–5.4.4
- scDF Overall multiplicative scaling factor applied to dose factors of all elements in structure [optional]

Note

Normally, **Raith_structure.plotedges** is called without arguments. The optional arguments M and scDF are used internally, when **Raith_structure.plotedges** is called by **Raith_library.plotedges** or **Raith_positionlist.plotedges**.

Calling **Raith_structure.plotedges** does not change the current axis scaling; issue an `axis equal` command to ensure that the element is displayed in the figure correctly.

Example

Given the **Raith_structure** object s defined in the above “**Constructor**” section:

```
s.plotedges;
axis equal;
```

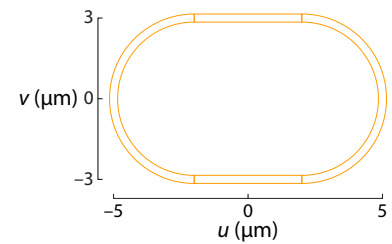


Figure 4.2: Racetrack resonator structure plotted using the **Raith_structure.plotedges** method

5

The Raith_library class

Class overview: Raith_library

Properties (public)

name String specifying name of GDSII library
structures Array of Raith_structure objects in library

Properties (private set access)

structlist Cell array of all structure names in library

Methods

append Append Raith_structure object(s) to library
writegds Output Raith GDSII hierarchy (.csf) file
plot Plot structure in library as filled polygons
plotedges Plot structure in library as unfilled polygons

Static Methods

trans Return augmented matrix for translation
rot Return augmented matrix for rotation
refl Return augmented matrix for reflection about u -axis
scale Return augmented matrix for uniform scaling
writerec Write GDSII record
writehead Write GDSII library header records
writeelement Write GDSII element record(s)
writebeginstruct Write GDSII records to begin a structure
writeendstruct Write GDSII record to end a structure
writeendlib Write GDSII record to end a library

RAITH_LIBRARY OBJECTS define GDSII hierarchies containing collections of structures (Raith_structure objects) which may be referred to in positionlist entries. The **Raith_library.writegds** method outputs a “Raith-dialect” GDSII (.csf) file which can be used by the Raith beamwriting software without any additional modification. Additionally, if all referenced structures are contained in the li-

brary, the full hierarchy of structures containing 'sref' or 'aref' elements may be displayed using the **Raith_library.plot** and **Raith_library.plotedges** methods.

5.1 Properties

5.1.1 Public properties

name: String specifying name of GDSII library, not including .csf extension.

structures: Array of Raith_structure objects in library. Raith_structure objects may be added to **structures** either using standard MATLAB notation, or via the **Raith_library.append** method.

5.1.2 Private set-access properties

structlist: Ordered cell array of all names of structures found in library. **structlist** is automatically updated whenever **structures** is amended.

5.2 Constructor

```
L=Raith_library(name,structures)
```

By default, all properties are checked for correctness (typing, allowed values, size) before being assigned, whether the Raith_library object is created with a constructor or its properties are amended individually.

Arguments

name String specifying name of GDSII library, not including .csf extension.

structures Array of Raith_structure objects in library. Raith_structure objects may be added to **structures** either using standard MATLAB notation, or via the **Raith_library.append** method.

Example

Given the Raith_structure object *s* defined in §4.2:

```
% Racetrack resonator defined in Raith_structure object S
lbl=Raith_structure('radius_label',Raith_element('text',0, ...
[0 0],2,0,[1 0],'3 um',1.5));
L=Raith_library('resonators',[S lbl]);
```

5.3 Methods

5.3.1 ***Raith_library.append** method*

Description

Append Raith_structure object(s) to library; structure names are checked for uniqueness.

Syntax

```
append(S)
```

Arguments

S Raith_structure object (or array thereof) to be appended to library

Example

Given the Raith_structure objects *S* and *lbl*, defined in §4.2 and the above “[Constructor](#)” section, respectively, the three following commands all yield the same library *L*:

```
% Using Raith_library.append
L=Raith_library('resonators',S);
L.append(lbl);

% Using horizontal concatenation
L=Raith_library('resonators',[S lbl]);

% Using array indexing
L=Raith_library('resonators',S);
L.structures(end+1)=lbl;
```

5.3.2 ***Raith_library.writegds** method*

Description

Write Raith GDSII hierarchy of all structures to file *<name>.csf*

Syntax

```
writegds
writegds(outdir)
```

Arguments

outdir String specifying directory in which to write .csf file [optional]; if called without arguments, file is written to working directory.

Example

Given the Raith_library object `L` in the above “Constructor” section:

```
L.writegds('C:\Users\Public\Documents');

Checking for missing structures...OK.
Writing C:\Users\Public\Documents\resonators.csf...
  Header information
  Structure 1/2: racetrack
  Structure 2/2: radius_label
GDSII library resonators.csf successfully written.
```

5.3.3 *Raith_library.plot* method

Description

Plot structure in library with Raith dose factor colouring. Elements are displayed as filled polygons, where applicable ('polygon'; 'path' with non-zero **data.w**; 'arc', 'circle', and 'ellipse' with empty **data.w**; 'text'). All elements in the structure are plotted, regardless of **data.layer** value. The full hierarchy of structures including 'sref' or 'aref' elements are displayed if all structures being referenced are present in the library.

Syntax

```
plot(structname)
plot(structname,M)
plot(structname,M,scDF)
```

Arguments

- structname** String specifying name of structure to be plotted (must be in **structlist**)
- M** Augmented transformation matrix to be applied to structure [optional]; see §§5.4.1–5.4.4
- scDF** Overall multiplicative scaling factor applied to dose factors of all elements in structure [optional]

Note

Normally, **Raith_library.plot** is called without arguments, to display the structure as it would appear in the Raith software. The optional arguments **M** and **scDF** are used internally, when **Raith_library.plot** is called by **Raith_positionlist.plot**.

Calling **Raith_library.plot** does not change the current axis scaling; issue an `axis equal` command to ensure that the element is displayed in the figure correctly.

Example

Given the Raith_structure objects `s` and `lbl`, defined in §4.2 and §5.2, respectively:

```
% Racetrack resonator defined in Raith_structure object S
% Radius label defined in Raith_structure object lbl
E(1)=Raith_element('sref','racetrack',[0 0]);
E(2)=Raith_element('sref','radius_label',[0 -4]);
RR=Raith_structure('labelled_racetrack',E);

L=Raith_library('resonators',RR);
L.plot('labelled_racetrack'); % Figure 5.1

L.append(S);
clf;
L.plot('labelled_racetrack'); % Figure 5.2
axis equal;

L.append(lbl);
clf;
L.plot('labelled_racetrack'); % Figure 5.3
axis equal;
```

5.3.4 Raith_library.plotedges method

Description

Plot outlines of structure in library with Raith dose factor colouring. Elements are displayed as unfilled polygons, where applicable ('polygon'; 'path' with non-zero `data.w`; 'arc', 'circle', and 'ellipse' with empty `data.w`; 'text'). All elements in the structure are plotted, regardless of `data.layer` value. The full hierarchy of structures including 'sref' or 'aref' elements are displayed if all structures being referenced are present in the library.

Syntax

```
plotedges(structname)
plotedges(structname,M)
plotedges(structname,M,scDF)
```

Arguments

- structname** String specifying name of structure to be plotted (must be in **structlist**)
- M** Augmented transformation matrix to be applied to structure [optional]; see §§5.4.1–5.4.4
- scDF** Overall multiplicative scaling factor applied to dose factors of all elements in structure [optional]

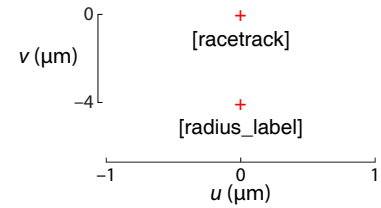


Figure 5.1: Display resulting from **Raith_library.plot** method when referenced structures are not in library

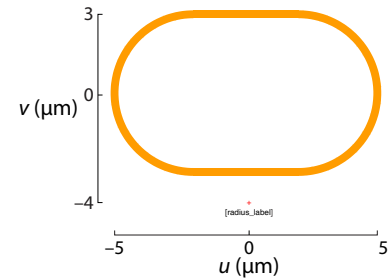


Figure 5.2: Display resulting from **Raith_library.plot** method when one referenced structure is not in library

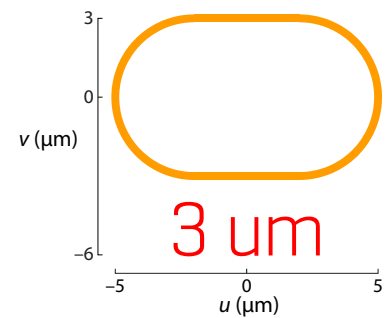


Figure 5.3: Display resulting from **Raith_library.plot** method when all structures are present in library

Note

Normally, **Raith_library.plotedges** is called without arguments, to display the structure as it would appear in the Raith software. The optional arguments **M** and **scDF** are used internally, when **Raith_library.plotedges** is called by **Raith_positionlist.plotedges**. Calling **Raith_library.plotedges** does not change the current axis scaling; issue an `axis equal` command to ensure that the element is displayed in the figure correctly.

Example

Given the **Raith_library** object **L** defined in the final example of §5.3.3:

```
L.plotedges('labelled_racetrack');
axis equal;
```

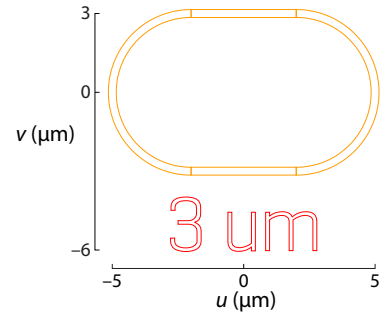


Figure 5.4: Display resulting from **Raith_library.plotedges** method when all structures are present in library

5.4 Static Methods

THE METHODS IN THIS SECTION do not require an instance of the **Raith_library** class to be called (static), and are generally used internally. Certain circumstances, however, may require the user to call them explicitly (e.g., see §7.3).

5.4.1 Raith_library.trans method**Description**

Return augmented matrix for translation.

Syntax

```
M=Raith_library.trans(p)
```

Arguments

p Translation vector; 1×2 vector $[p_u \ p_v]$ (μm)

Return value

M Augmented matrix for translation

Note

For translation by a vector \vec{p} , the augmented matrix is

$$\begin{bmatrix} 1 & 0 & p_u \\ 0 & 1 & p_v \\ 0 & 0 & 1 \end{bmatrix} \quad (5.1)$$

Example

```
Raith_library.trans([10 20])

ans =

    1     0    10
    0     1    20
    0     0     1
```

5.4.2 *Raith_library.rot* method**Description**

Return augmented matrix for rotation.

Syntax

```
M=Raith_library.rot(theta)
```

Arguments

theta Rotation angle, counter-clockwise positive (degrees)

Return value

M Augmented matrix for rotation

Note

For counter-clockwise rotation through an angle θ , the augmented matrix is

$$\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.2)$$

Example

```
Raith_library.rot(30)

ans =

    0.8660   -0.5000     0
    0.5000    0.8660     0
         0         0    1.0000
```

5.4.3 *Raith_library.refl* method**Description**

Return augmented matrix for reflection about u -axis n times.

Syntax

```
M=Raith_library.refl(n)
```

Arguments

n Number of times to reflect about u -axis

Return value

M Augmented matrix for reflection

Note

For reflection about the u -axis n times, the augmented matrix is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & (-1)^n & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.3)$$

Example

```
Raith_library.refl(1)
```

```
ans =
```

```

1      0      0
0     -1      0
0      0      1
```

5.4.4 *Raith_library.scale* method**Description**

Return augmented matrix for uniform scaling

Syntax

```
M=Raith_library.scale(mag)
```

Arguments

mag Uniform scaling factor

Return value

M Augmented matrix for uniform scaling

Note

For uniform scaling by a factor m , the augmented matrix is

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.4)$$

Example

```
Raith_library.scale(3)

ans =

     3     0     0
     0     3     0
     0     0     3
```

5.4.5 *Raith_library.writerec* method

Description

Write single GDSII record to file

Syntax

```
Raith_library.writerec(FileID, rectype, datatype, parameters)
```

Arguments

- FileID** Integer file identifier obtained from MATLAB's **fopen** function
- rectype** GDSII record type, specified in decimal format; Table 5.1 lists the record types used in the Raith_GDSII toolbox.
- datatype** GDSII data type, specified in decimal format; Table 5.2 lists the data types for the GSDII specification.
- parameters** Record parameters, of type defined by datatype

Example

```
% Open a file for writing
FileID=fopen('test.csf','w');
% Write a BOUNDARY record, which contains no data
Raith_library.writerec(8,0,[]);
```

5.4.6 *Raith_library.writehead* method

Description

Write GDSII library header records

Syntax

```
Raith_library.writehead(FileID,name)
```

Arguments

- FileID** Integer file identifier obtained from MATLAB's **fopen** function
- rectype** GDSII library name, without .csf extension (string)

Record type	Hex	Dec
HEADER	0x00	0
BGNLIB	0x01	1
LIBNAME	0x02	2
UNITS	0x03	3
ENDLIB	0x04	4
BGNSTR	0x05	5
STRNAME	0x06	6
ENDSTR	0x07	7
BOUNDARY	0x08	8
PATH	0x09	9
SREF	0x0A	10
AREF	0x0B	11
LAYER	0x0D	13
DATATYPE	0x0E	14
WIDTH	0x0F	15
XY	0x10	16
SNAME	0x12	18
COLROW	0x13	19
STRANS	0x1A	26
MAG	0x1B	27
ANGLE	0x1C	28
CURVED ¹	0x56	86

Table 5.1: GDSII record types, with values in hexadecimal and decimal format. The latter is passed to **Raith_library.writerec** as the **rectype** argument.

¹ The Raith CURVED element record type is not part of the GDSII specification. It is used by the Raith software to denote arc, ellipse, and circle elements.

Data type	Hex	Dec
No data present	0x00	0
Bit array (2 bytes)	0x01	1
2-byte signed integer	0x02	2
4-byte signed integer	0x03	3
4-byte float ²	0x04	4
8-byte float	0x05	5
ASCII string	0x06	6

Table 5.2: GDSII data types, with values in hexadecimal and decimal format. The latter is passed to **Raith_library.writerec** as the **datatype** argument.

² The 4-byte float data type is listed as unused in the GDSII Stream Format Manual v6.0.

Note

This method writes the HEADER, BGNLIB, LIBNAME, and UNITS records. The current system time is used for the BGNLIB record, a “.csf” is appended to name for the LIBNAME record, and 1 μm and 1 nm are used for the user and database units, respectively, in the UNITS record.

Example

```
% Open a file for writing
FileID=fopen('test.csf','w');
% Write GDSII header information
Raith_library.writehead(FileID,'test');
```

5.4.7 *Raith_library.writebeginstruct* method**Description**

Write GDSII records to begin a structure

Syntax

```
Raith_library.writebeginstruct(FileID,name)
```

Arguments

FileID Integer file identifier obtained from MATLAB's **fopen** function
name Name of structure (string)

Note

This method writes the BGNSTR and STRNAME records. The current system time is used for BGNSTR.

Example

```
% Open a file for writing
FileID=fopen('test.csf','w');
Raith_library.writebeginstruct(FileID,'waveguide');
```

5.4.8 *Raith_library.writeelement* method**Description**

Write GDSII element records

Syntax

```
Raith_library.writeelement(FileID,element)
```

Arguments

FileID Integer file identifier obtained from MATLAB's **fopen** function

element Raith_element object describing element

Note

The GDSII record types written vary according to the type of element.

Example

```
% Open a file for writing
FileID=fopen('test.csf','w');
% Define an element
E=Raith_element('path',0,[0 1 1;0 0 1],0.2,1);
Raith_library.writeelement(FileID,E);
```

5.4.9 Raith_library.writeendstruct method**Description**

Write GDSII record to end a structure

Syntax

```
Raith_library.writeendstruct(FileID)
```

Arguments

FileID Integer file identifier obtained from MATLAB's **fopen** function

Note

This method writes the ENDSTR record, which has no parameters.

Example

```
% Open a file for writing
FileID=fopen('test.csf','w');
Raith_library.writeendstruct(FileID);
```

5.4.10 Raith_library.writeendlib method**Description**

Write GDSII record to end a library

Syntax

```
Raith_library.writeendlib(FileID)
```

Arguments

FileID Integer file identifier obtained from MATLAB's **fopen** function

Note

This method writes the ENDLIB record, which has no parameters.

Example

```
% Open a file for writing
FileID=fopen('test.csf','w');
Raith_library.writeendlib(FileID);
```

6

The Raith_positionlist class

Class overview: Raith_positionlist

Properties (public)

library Raith_library object containing all structures
csf_path Path of GDSII library on Raith computer
WF Writefield dimensions
chipUV Size of rectangular chip (specimen)
poslist Structure array of positionlist entries

Methods

append Append Raith_structure object to positionlist
plot Plot entire positionlist as filled polygons
plotedges Plot entire positionlist as unfilled polygons
plotWA Plot working areas of all structures in positionlist
plotWF Plot first writefields of all structures in positionlist
centre Centre current positionlist entries on chip
writepls Output Raith positionlist (.pls) file

RAITH_POSITIONLIST OBJECTS define positionlists: a sequential list of instructions to write a structure defined in a GDSII hierarchy at a certain position on the chip. The **Raith_positionlist.writepls** method outputs a Raith positionlist (.pls) file which can be used by the Raith beamwriting software without any additional modification. As an aid to chip layout, the structures, working areas, and writefields of the entire positionlist can be plotted.

6.1 *Properties*

library: Raith_library object containing all structures to be referenced in positionlist
csf_path: Full path of GDSII hierarchy (.csf) file, as found on the Raith computer

- WF**: Writefield size; 1×2 vector [$size_u$ $size_v$] (μm)
- chipUV**: Size of rectangular chip; 1×2 vector [$size_u$ $size_v$] (mm)
- poslist**: Structure array of positionlist entries, containing fields **name**, **uv_c**, **DF**, **WA**, and **layers**; see 6.3.1 for a description of these fields

6.2 Constructor

```
P=Raith_positionlist(library,csf_path,WF,chipUV)
```

By default, the arguments are checked for correctness (typing, allowed values, size) before being assigned, whether the `Raith_positionlist` object is created with a constructor or these four properties are amended individually.

Arguments

- library** `Raith_library` object containing all structures to be placed in positionlist
- csf_path** Full path of GDSII hierarchy file, as found on Raith computer
- WF** Writefield size; 1×2 vector [$size_u$ $size_v$] (μm)
- chipUV** Size of rectangular chip; 1×2 vector [$size_u$ $size_v$] (mm)

Example

Given the `Raith_structure` object `s` defined in §4.2:

```
% Racetrack resonator defined in Raith_structure object S
lbl=Raith_structure('radius_label',Raith_element('text',0, ...
    [0 0],2,0,[1 0],'3 um',1.5));
L=Raith_library('resonators',[S lbl]);
% Positionlist for a 5 mm x 5 mm chip, 100 um writefield
P=Raith_positionlist(L,'F:\Raith\resonators.csf', ...
    [100 100],[5 5]);
```

6.3 Methods

6.3.1 *Raith_positionlist.append* method

Description

Append `Raith_structure` object to positionlist.

Syntax

```
append(name,uv_c,DF,WA)
append(name,uv_c,DF,WA, layers)
```

Arguments

- name** Raith_structure object name (as found in GDSII library)
- uv_c** Centre of first writefield of structure; 1×2 vector $[u_c \ v_c]$ (mm)
- DF** Overall dose factor scaling for entire structure
- WA** Working area of structure; 1×4 vector $[u_{\min} \ v_{\min} \ u_{\max} \ v_{\max}]$ (μm)
- layers** Vector of layers to expose [optional]; defaults to all layers present in structure elements

Note

The units of the arguments reflect how they are stored in the positionlist: **uv_c** is in mm and **WA** is in μm . The working area coordinates are defined with respect to the origin of the Raith_structure object. As in the Raith software, the bottom-left corners of the working area and the writefield are aligned; as such, **uv_c** specifies a point half a writefield width above and to the right of the bottom-left corner of the working area. One design paradigm which simplifies the positioning of structures contained in a single writefield is to define them centred on the origin, then specify a working area—also centred on the origin—which is the same size as the writefield (see following Example). In this case, **uv_c** will correspond to the origin of the structure.

Example

Given the Raith_library and Raith_positionlist objects **L** and **P**, respectively, defined in the above “[Constructor](#)” section:

```
% Write a racetrack resonator and label near the top-left
% corner of the chip (100 um WF)
P.append('racetrack',[1 4],1,[-50 -50 50 50]);
P.append('radius_label',[1 3.996],1,[-50 -50 50 50]);
```

6.3.2 Raith_positionlist.plot method

Description

Plot all structures in positionlist with Raith dose factor colouring, with chip outline. Elements are displayed as filled polygons, where applicable ('polygon'; 'path' with non-zero **data.w**; 'arc', 'circle', and 'ellipse' with empty **data.w**; 'text'). All elements in the structures are plotted, regardless of **data.layer** value. The full hierarchy of structures including 'sref' or 'aref' elements are displayed if all structures being referenced are

present in the library contained in the `Raith_positionlist` object.

Syntax

`plot`

Arguments

(None)

Note

Calling **`Raith_positionlist.plot`** sets the axis scaling to equal; all plotted objects therefore appear with correct scaling. The axes are in units of μm .

Example

Given the `Raith_positionlist` object defined in §6.3.1, above:

```
P.plot; % Structures are too small to see at this scale
axis([990 1010 3990 4010]); % Zoom to structures
```

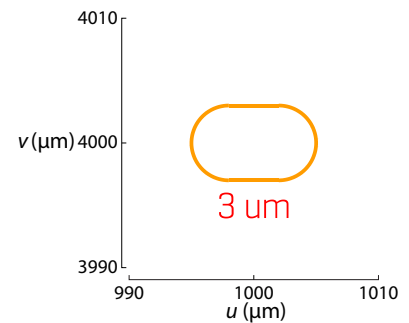


Figure 6.1: Positionlist plotted using the **`Raith_positionlist.plot`** method

6.3.3 ***`Raith_positionlist.plotedges`** method*

Description

Plot outlines of all structures in positionlist with Raith dose factor colouring, with chip outline. Elements are displayed as unfilled polygons, where applicable ('**`polygon`**'; '**`path`**' with non-zero **`data.w`**; '**`arc`**', '**`circle`**', and '**`ellipse`**' with empty **`data.w`**; '**`text`**'). All elements in the structures are plotted, regardless of **`data.layer`** value. The full hierarchy of structures including '**`sref`**' or '**`aref`**' elements are displayed if all structures being referenced are present in the library contained in the `Raith_positionlist` object.

Syntax

`plotedges`

Arguments

(None)

Note

Calling **`Raith_positionlist.plotedges`** sets the axis scaling to equal; all plotted objects therefore appear with correct scaling. The axes are in units of μm .

Example

Given the `Raith_positionlist` object defined in §6.3.1, above:

```
P.plotedges; % Structures are too small to see at this scale
axis([990 1010 3990 4010]); % Zoom to structures
```

6.3.4 *Raith_positionlist.plotWA* method

Description

Plot working area of all structures in positionlist in dotted blue lines, with chip outline.

Syntax

```
plotWA
```

Arguments

(None)

Note

Calling **`Raith_positionlist.plotWA`** sets the axis scaling to equal; all working areas therefore appear with correct scaling.

Example

To illustrate the alignment of working areas and writefields, this example specifies working areas which are smaller than the writefield. Assume the `Raith_library` and `Raith_positionlist` objects `L` and `P`, respectively, are defined as in §6.2; to preserve the relative alignment of the racetrack and the label, `uv_c` must be changed to accommodate the difference in WA:

```
P.append('racetrack',[1 4],1,[-10 -10 20 10]);
P.append('radius_label',[0.99 4.001],1,[-20 -5 10 5]);
P.plot; % Plot structures
P.plotWA; % Plot working areas
axis([935 985 3945 3975]); % Zoom to structures
```

6.3.5 *Raith_positionlist.plotWF* method

Description

Plot writefields of all structures in positionlist in dotted green lines; writefield centres are marked with a + sign.

Syntax

```
plotWA
```

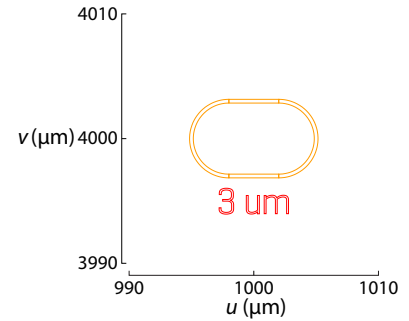


Figure 6.2: Positionlist plotted using the **`Raith_positionlist.plotedges`** method

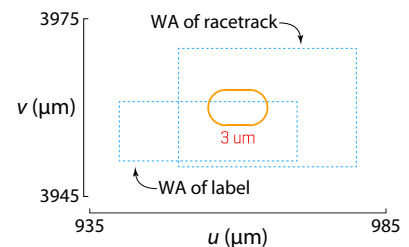


Figure 6.3: Working areas in positionlist plotted using the **`Raith_positionlist.plotWA`** method

Arguments

(None)

Note

Calling **Raith_positionlist.plotWF** sets the axis scaling to equal; all working areas therefore appear with correct scaling.

Example

Given all objects defined as in the above “**Raith_positionlist.plotWA method**” Example:

```
P.plot; % Plot structures
P.plotWA; % Plot working areas
P.plotWF; % Plot writefields
axis([940 1050 3950 4055]); % Zoom to structures
```

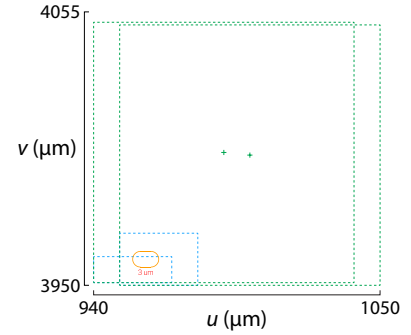


Figure 6.4: Working areas and writefields in positionlist plotted using the **Raith_positionlist.plotWA** and **Raith_positionlist.plotWF** methods

6.3.6 Raith_positionlist.centre method**Description**

Centre current positionlist entries on the chip, preserving relative spacing, with the option of matrix-copying them.

Syntax

```
centre
centre(mbyn)
```

Arguments

mbyn Number of rows and columns of the matrix of sub-chips [optional]; 1×2 vector $[m \ n]$

Note

If called with no argument, the current positionlist entries are shifted such that the overall pattern (as defined by the working areas) are centred both vertically and horizontally on the chip. If called with the optional **mbyn** argument, the chip is divided into an m -by- n matrix of equal-sized rectangular sub-chips, and the positionlist entries are centred as described above in each sub-chip. In either case, the **poslist** property is rewritten; there is no built-in way of undoing this operation.

Example

Given the **Raith_positionlist** object defined in §6.3.4, above:

```
P.plotWF; % Before centring
P.centre;
```

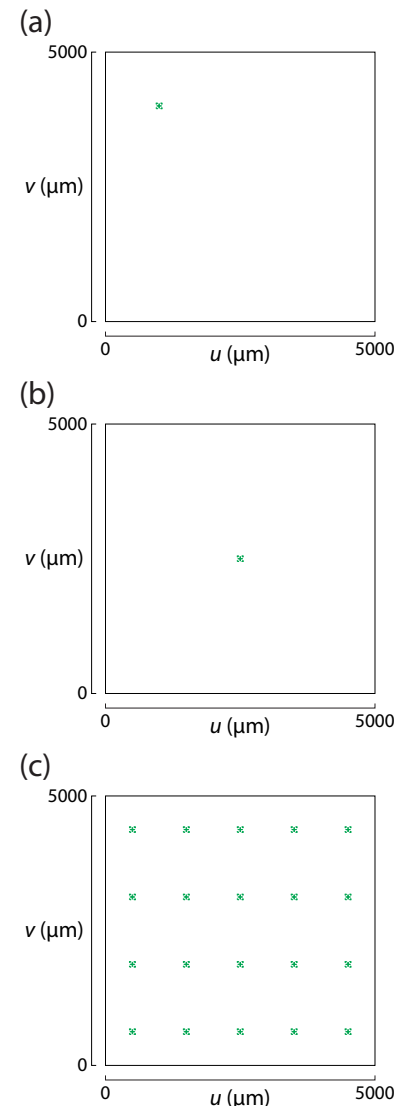


Figure 6.5: Position of writefields (a) before centring, (b) after issuing **P.centre**, and (c) after issuing **P.centre([4 5])**

```
P.plotWF; % After centring
P.centre([4 5]);
P.plotWF; % After matrix-copying
```

6.3.7 *Raith_positionlist.writepls* method

Description

Syntax

```
writepls
writepls(filepath)
```

Arguments

filepath Full path of positionlist file to be written, including .pls extension [optional]

Note

If called without argument, a **<library.name>.pls** file is written to the current directory.

Example

Given the *Raith_positionlist* object defined in §6.3.4, above:

```
P.writepls;

Writing C:\Users\Public\Documents\resonators.pls...
Positionlist resonators.pls successfully written.
```


7

Extended techniques

THE NORMAL USAGE of the Raith_GDSII toolbox is explained in the foregoing sections of this document. In this section, we describe extended techniques which may be used to save time and/or memory—two concerns which can become limiting factors when generating patterns consisting of very large numbers of elements. The cost of these extended techniques includes the loss of Raith_element, Raith_structure, and Raith_library data checking and all plotting functionality, so they are best used in the final stages of the design process.

7.1 Disabling data checking

BY DEFAULT, the properties of Raith_element, Raith_structure, and Raith_library objects are checked for correctness (typing, allowed values, size) before being assigned or altered, ensuring that the resulting objects may be plotted and used to generate GDSII and positionlist files without subsequent errors. By disabling data checking, Raith_element, Raith_structure, and Raith_library objects may be created more quickly, at the risk of encountering more cryptic error messages downstream.

Data checking may be disabled by declaring a global variable `checkdata` and assigning it a value of logical zero (Boolean `false`).

Example

Creating a series of random Raith_element 'path' objects with and without data checking, resulting in a greater than 4× reduction in computation time:

```
% With data checking
tic;
for k=1:1000
E=Raith_element('path',0,rand(2,1000),0,1);
end
t_Check=toc;
```

```

% Without data checking
global checkdata
checkdata=false;
tic;
for k=1:1000
E=Raith_element('path',0,rand(2,1000),0,1);
end
t_noCheck=toc;

t_Check/t_noCheck

ans =

    4.4510

```

Note

As a global variable, `checkdata` will remain accessible to any functions declaring it as global, even if a `clear` command is issued in the current workspace. To remove `checkdata` completely (i.e., restoring the default state of enabled data checking), use `clear global checkdata`.

7.2 Defining patterns using MATLAB structures

THE TIME AND MEMORY OVERHEAD associated with creating `Raith_element` and `Raith_structure` objects can be further reduced—at the expense of losing all plotting functionality—by using MATLAB structures¹ with fields that match the `Raith_element` and `Raith_structure` properties, both in name and in type. “Elements” and “structures” defined in this way can be used by `Raith_library` and `Raith_positionlist` objects to generate .csf and .pls files, provided data checking is disabled (see §7.1, above). Refer to §3 and §4 for descriptions of `Raith_element` and `Raith_structure` class properties.

¹ That is, instances of the built-in `struct` class. See the MATLAB User Guide ► Programming Fundamentals ► Classes (Data Types) ► Structures.

Example

Creating an array of microrings with varying widths:

```

% Cell array of ring widths (um)
w=num2cell(0.1:0.05:0.5)';

% Cell array of disk centres: 15 um spacing in u, v=0
uv_c=num2cell(15*(0:length(w)-1)'+[1 0],2);

% 'circle' Raith_element objects have properties
% 'type', and 'data', the latter having fields 'layer',
% 'uv_c', 'r', 'w', 'N', and 'DF', so we create a struct
% array with these fields
data=struct('layer',0,'uv_c',uv_c,'r',5,'w',w,'N',100,'DF',1.3);
E=struct('type','circle','data',num2cell(data));

% Raith_structure objects have properties

```

```

% 'name', 'elements', and 'reflist'
S.name='rings';
S.elements=E;
S.reflist=[];

global checkdata;
checkdata=false;
L=Raith_library('ringarray',S);
L.writegds;

Skipping all data checking.
Writing C:\Users\Public\Documents\ringarray.csf...
    Header information
    Structure 1/1: rings
GDSII library ringarray.csf successfully written.

```

Note

In the above Example, the `data` and `E` variables were created using MATLAB's **struct** function, which allows a structure array to be output if any of the value inputs are cell arrays; the same result could have been produced using a `for` loop.

7.3 “On-the-fly” GDSII writing

IN SITUATIONS WHERE the number of elements in the pattern is so large that there is insufficient memory to keep them all in the MATLAB workspace simultaneously, it is still possible to output a GDSII file by generating elements and writing them sequentially; this procedure uses the static `Raith_library` methods described in §§5.4.6–5.4.10 to write GDSII records directly². The following general format must be observed:

```

Raith_library.writehead (Header information)
    Raith_library.writebeginstruct (Begin first structure in library)
        Raith_library.writeelement (First element in structure)
        :
        Raith_library.writeelement (Last element in structure)
    Raith_library.writeendstruct (End first structure)
    :
    Raith_library.writebeginstruct (Begin last structure in library)
        Raith_library.writeelement (First element in structure)
        :
        Raith_library.writeelement (Last element in structure)
    Raith_library.writeendstruct (End last structure)
Raith_library.writeendlib (End library)

```

² Hard-core enthusiasts should note that it is possible to write an entire arbitrary GDSII file using only the **Raith_library.writegds** method; such a feat is beyond the scope of this User Guide.

When writing GDSII files “on the fly”, the user is responsible for ensuring that all structures referenced by `'sref'` and `'aref'` elements are in fact present in the library.

Example

Creating the same array of microrings as in the above Example

§7.2 “on the fly”:

```
% Vector of ring widths (um)
w=0.1:0.05:0.5;

% Matrix of disk centres: 15 um spacing in u, v=0
uv_c=15*(0:length(w)-1)*[1 0];

% Open file for writing binary data
FileID=fopen('C:\Users\Public\Documents\ringarray.csf','w');

Raith_library.writehead(FileID,'ringarray');
Raith_library.writebeginstruct(FileID,'rings');

% Loop through all elements in structure; only one
% Raith_element object is in memory at any given time.
for k=1:length(w)
E=Raith_element('circle',0,uv_c(k,:),5,w(k),100,1.3);
Raith_library.writeelement(FileID,E);
end

Raith_library.writeendstruct(FileID);
Raith_library.writeendlib(FileID);

fclose(FileID);
```

Note

In the above Example, the `data` and `E` variables were created using MATLAB’s **struct** function, which allows a structure array to be output if any of the value inputs are cell arrays; the same result could have been produced using a `for` loop.

Index

- .asc format, 9
- .csf file, 10, 33, 45
- .elm format, 9
- .pls file, 10, 45

- absolute magnification, 10
- absolute rotation, 10
- arc element, 10, 17, 20, 21, 27, 30, 31, 36, 37, 47, 48
- aref element, 10, 17, 19, 26, 30, 34, 36, 37, 47, 48, 55

- bug reporting, 11

- circle element, 10, 17, 21, 22, 27, 30, 31, 36, 37, 47, 48
- citations, 11
- computation speed, 53, 54
- constructors
 - Raith_element, 17
 - Raith_library, 34–36
 - Raith_positionlist, 46, 47
 - Raith_structure, 30–32
- curved element, 9, 10, 20–22

- data checking, disabling, 53
- dot element, 10, 17, 20
- download, 11

- element types
 - arc, 10, 17, 20, 21, 27, 30, 31, 36, 37, 47, 48
 - aref, 10, 17, 19, 26, 30, 34, 36, 37, 47, 48, 55
 - circle, 10, 17, 21, 22, 27, 30, 31, 36, 37, 47, 48
 - dot, 10, 17, 20
 - ellipse, 10, 22, 27, 30, 31, 36, 37, 47, 48
 - path, 10, 17, 19, 27, 30, 31, 36, 37, 47, 48, 53
 - polygon, 10, 13, 17, 18, 27, 30, 31, 36, 37, 47, 48
 - sref, 10, 17, 19, 25, 30, 34, 36, 37, 47, 48, 55
 - text, 10, 17, 23, 24, 27, 30, 31, 36, 37, 47, 48
- ellipse element, 10, 22, 27, 30, 31, 36, 37, 47, 48

- installation, 11

- memory
 - inadequate, 53–55
- methods
 - append** (Raith_library), 34, 35
 - append** (Raith_positionlist), 46
 - centre** (Raith_positionlist), 50
 - plot** (Raith_element), 25–27
 - plot** (Raith_library), 25–27, 31, 34, 36, 37
 - plot** (Raith_positionlist), 27, 31, 36, 47, 48
 - plot** (Raith_structure), 27, 30, 31
 - plotedges** (Raith_element), 27, 28
 - plotedges** (Raith_library), 28, 32, 34, 37, 38
 - plotedges** (Raith_positionlist), 28, 32, 38, 48, 49
 - plotedges** (Raith_structure), 28, 31, 32
 - plotWA** (Raith_positionlist), 49, 50
 - plotWF** (Raith_positionlist), 49, 50
 - refl** (Raith_library), 39
 - rot** (Raith_library), 39
 - scale** (Raith_library), 40
 - trans** (Raith_library), 38
 - writebeginstruct** (Raith_library), 42, 55
 - writteelement** (Raith_library), 42, 55
 - writeendlib** (Raith_library), 43, 55
 - writeendstruct** (Raith_library), 43, 55
 - writedges** (Raith_library), 33, 35
 - writehead** (Raith_library), 41, 55
 - writelps** (Raith_positionlist), 45, 51
 - writerec** (Raith_library), 41, 55

- “on-the-fly” GDSII writing, 55

- path element, 10, 17, 19, 27, 30, 31, 36, 37, 47, 48, 53
- polygon element, 10, 13, 17, 18, 27, 30, 31, 36, 37, 47, 48

- properties
 - chipUV** (Raith_positionlist), 45
 - csf_path** (Raith_positionlist), 45
 - data** (Raith_element), 17, 18
 - data.angle** (Raith_element), 20, 22, 25, 26
 - data.DF** (Raith_element), 20, 27, 28
 - data.layer** (Raith_element), 30, 31, 36, 37, 47, 48
 - data.mag** (Raith_element), 25, 26
 - data.N** (Raith_element), 21
 - data.name** (Raith_element), 25, 26
 - data.reflect** (Raith_element), 25, 26
 - data.theta** (Raith_element), 21
 - data.uv** (Raith_element), 18, 20
 - data.uv_0** (Raith_element), 25
 - data.w** (Raith_element), 27, 30, 31,

- 36, 37, 47, 48
 - elements** (Raith_structure), 29, 30
 - library** (Raith_positionlist), 45, 51
 - name** (Raith_library), 33, 35
 - name** (Raith_structure), 29
 - poslist** (Raith_positionlist), 45, 50
 - reflist** (Raith_structure), 29, 30
 - structlist** (Raith_library), 33, 34, 36, 37
 - structures** (Raith_library), 33, 34
 - type** (Raith_element), 17, 18
 - WF** (Raith_positionlist), 45
- quick-start guide, 13
- Raith curved element, 9, 10, 20–22
- Raith_element class
- constructors, 17
 - methods
 - plot**, 25–27
 - plotedges**, 27, 28
 - overview, 17
 - properties
 - data**, 17, 18
 - data.angle**, 20, 22, 25, 26
 - data.DF**, 20, 27, 28
 - data.layer**, 30, 31, 36, 37, 47, 48
 - data.mag**, 25, 26
 - data.N**, 21
 - data.name**, 25, 26
 - data.reflect**, 25, 26
 - data.theta**, 21
 - data.uv**, 18, 20
 - data.uv_0**, 25
 - data.w**, 27, 30, 31, 36, 37, 47, 48
 - type**, 17, 18
- Raith_library class
- constructor, 34–36
 - methods
 - append**, 34, 35
 - plot**, 25–27, 31, 34, 36, 37
 - plotedges**, 28, 32, 34, 37, 38
 - refl**, 39
 - rot**, 39
 - scale**, 40
 - trans**, 38
 - writebeginstruct**, 42, 55
 - writteelement**, 42, 55
 - writeendlib**, 43, 55
 - writeendstruct**, 43, 55
 - writtegds**, 33, 35
 - writehead**, 41, 55
 - writerec**, 41, 55
 - overview, 33
 - properties
 - name**, 33, 35
 - structlist**, 33, 34, 36, 37
 - structures**, 33, 34
- Raith_positionlist class
- constructor, 46, 47
 - methods
 - append**, 46
 - centre**, 50
- plot**, 27, 31, 36, 47, 48
 - plotedges**, 28, 32, 38, 48, 49
 - plotWA**, 49, 50
 - plotWF**, 49, 50
 - writepls**, 45, 51
- overview, 45
- properties
- chipUV**, 45
 - csf_path**, 45
 - library**, 45, 51
 - poslist**, 45, 50
 - WF**, 45
- Raith_structure class
- constructor, 30–32
 - methods
 - plot**, 27, 30, 31
 - plotedges**, 28, 31, 32
 - overview, 29
 - properties
 - elements**, 29, 30
 - name**, 29
 - reflist**, 29, 30
- simply-connected font, 10, 23, 24
- sref element, 10, 17, 19, 25, 30, 34, 36, 37, 47, 48, 55
- text element, 10, 17, 23, 24, 27, 30, 31, 36, 37, 47, 48
- updates, 11
- user agreement, 11