TXLWizard Documentation

Release 1.7.0

Esteban Marin

Jun 07, 2016

INTRODUCTION

This document describes the usage and technical reference of the python program *TXLWizard* written by Esteban Marin (estebanmarin@gmx.ch).

1.1 What does it do?

The *TXLWizard* provides routines for generating TXL files (.txl) for the preparation of E-Beam lithography masks using python code. The TXL files can be processed with BEAMER. See the following links:

- http://genisys-gmbh.com/web/products/beamer.html

- http://cad035.psi.ch/LB_index.html

- http://cad035.psi.ch/LBDoc/BEAMER_Manual.pdf

The *TXLWizard* currently implements version *4.8* of the TextLIB (TXL) standard.

The generated TXL files are also converted to HTML / SVG for presentation in any modern browser or vector graphics application and allow rapid mask development.

Moreover, a command line interface *TXLConverter* provides conversion of existing TXL files to HTML / SVG (See Section *TXLConverter*).

1.2 Why should I use it?

TXL File Format:

- Text-based file format

- Can be generated with any scripting language (Python / Matlab / etc.)

- Easy to use

- Optimized E-Beam Performance due to *References* to objects and array of replicated objects (*SREF*, *AREF*)

TXLWizard:

- Create masks with well-structured scripts

- Flexible Python Scripting

- Mask-Code easy to read and reusable

- Automated label generation

1.3 Installation

The *TXLWizard* is written in python and will run in Python version 2.7+ and 3.1+.

In order to use it, the *TXLWizard* package must be available as a python package, i.e. either it must be copied to `Path_to_my_python_installation/site-packages/` or to the path where your script is located.

Alternatively, you can also prepend the following command to your python script:

```python
import sys
sys.path.append('path_to_the_folder_containing_TXLWizard')
```

Please note that this must be the parent folder containing the TXLWizard.

1.4 Structure / Pattern / Attribute

The following terms are used throughout this manual:

1.4.1 *Structure*

Refers to an object containing one or more *Pattern* objects. A *Structure* corresponds to the *STRUCT* command in TXL files.

1.4.2 *Pattern*

Refers to a pattern such as a circle, a polygon, an ellipse, a path, etc. The following patterns with the corresponding TXL command in brackets are supported:

- *Circle* (*C*)
- *Ellipse* (*ELP*)
- *Polygon* (*B*)
- *Polyline* (*P*)
- *Reference* (*SREF*)
- *Array* (*AREF*)

For more information, supported parameters, etc., see Section *Patterns*.

1.4.3 *Attribute*

Refers to an property of a *Pattern* determining the visual appearance of the *Pattern*. The following attributes with the corresponding TXL command in brackets are supported:

- *Layer* (*LAYER*)
- *DataType* (*DATATYPE*)
- *RotationAngle* (*ANGLE*)
- *StrokeWidth* (*WIDTH*)
- *ScaleFactor* (*MAG*)

Please note that the *TXLWizard* strictly implements the specification of the TXL format. This implies some peculiarities, such as

- *Attribute* commands preceed the corresponding *Pattern* in a *Structure* and are valid for all patterns that follow unless the attribute value is changed. Therefore, when adding a *Pattern* to a *Structure* with certain attributes, the attributes are valid for any subsequently added pattern, unless a different attribute value is specified.

- *Attribute* commands are valid for all patterns, except for *Reference* (*SREF*) and *Array* (*AREF*). Therefore the attributes of a pattern can only be specified in the structure where the pattern is added / defined.

- The *RotationAngle* attribute applies to each *Pattern* individually and rotates about each *Pattern*'s individual origin.

1.5 Example SVG Output

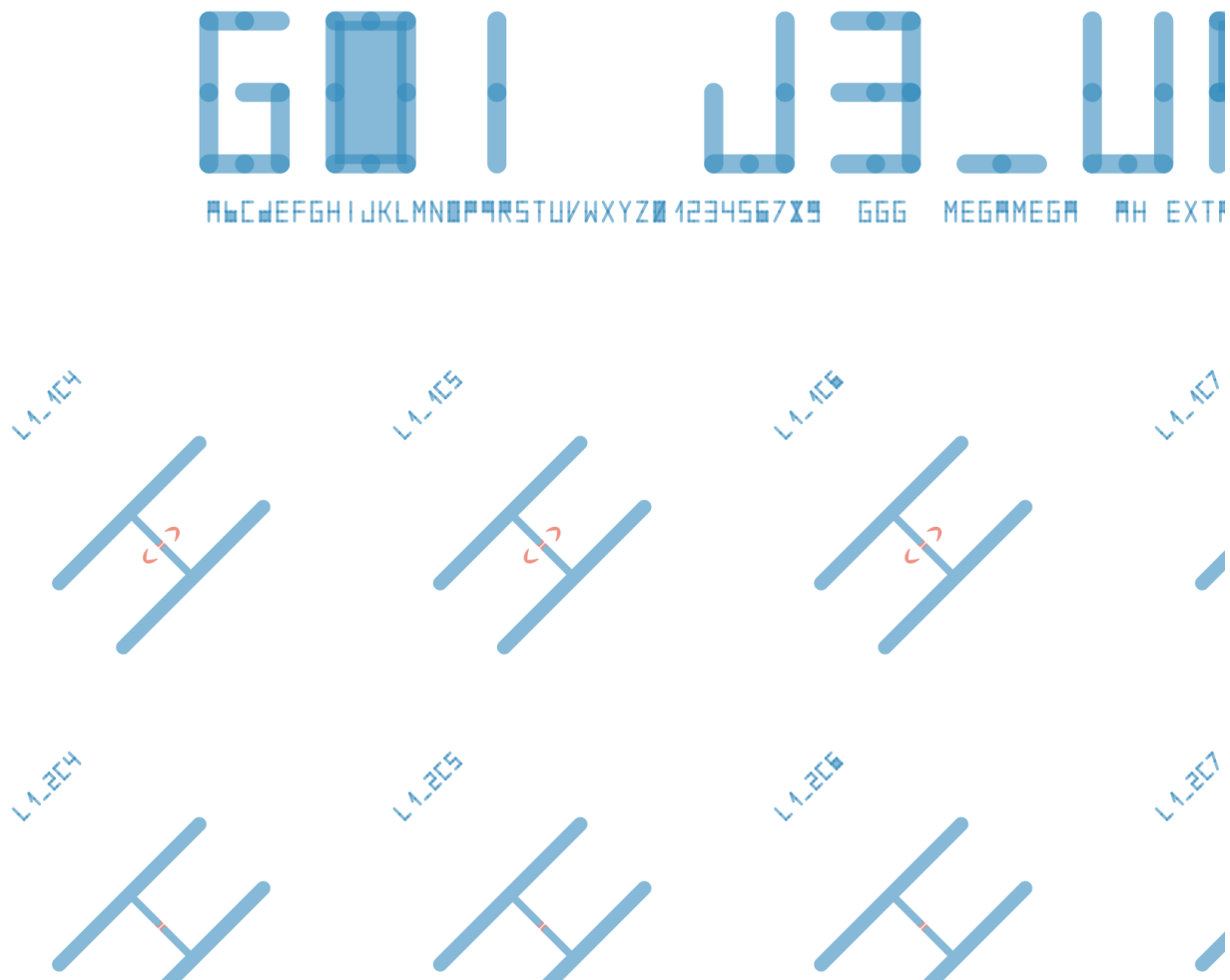An example output can be seen here:

Fig. 1.1: Example SVG output for a mask

1.6 How to start?

Have a look at the examples in Section *TXLWizard Examples* and consult the *Python Module Reference*. Happy scripting!

TXLWIZARD EXAMPLES

2.1 Introductory Example

2.1.1 Introduction

The following code demonstrates an introductory example usage of the *TXLWizard* for generating TXL files with python code.

The code can be found in the file Content/Example_Introduction.py. The resulting SVG image is shown in Figure *Generated SVG Image*.

Have a look at more advanced examples in Sections *Simple Example* and *Advanced Example* and at the *Python Module Reference*.

2.1.2 Code

```python
#############################################
# Import Libraries / Initialize TXLWriter #
#############################################

# Import TXLWriter, the main class for generating TXL Output
import TXLWizard.TXLWriter

# Import Pre-Defined Shapes / Structures wrapped in functions
import TXLWizard.ShapeLibrary.Label

# Initialize TXLWriter
TXLWriter = TXLWizard.TXLWriter.TXLWriter()

#####################
# Define Structures #
#####################

## Sample Label ##

# Give the sample a nice label
SampleLabelObject = TXLWizard.ShapeLibrary.Label.GetLabel(
    TXLWriter,
    Text='This is my text',
    OriginPoint=[-310, 240],
    FontSize=50,
    StrokeWidth=5,
    RoundCaps=True,  # Set to False to improve e-Beam performance
```

```
28        Layer=1
29    )
30
31    ## User Structure: Circle ##
32
33    # Create Content Structure for Circle with ID `MyCircle`
34    CircleStructure = TXLWriter.AddContentStructure('MyCircle')
35
36    # Add a `Pattern` of type `Circle`
37    CircleStructure.AddPattern(
38        'Circle',
39        Center=[0, 0],
40        Radius=150,
41        Layer=2
42    )
43
44    ##########################
45    # Generate Output Files #
46    ##########################
47
48    # Note: The suffix (.txl, .html, .svg) will be appended automatically
49    TXLWriter.GenerateFiles('Masks/Example_Introduction')
```

2.1.3 Generated SVG Image

2.2 Simple Example

2.2.1 Introduction

The following code demonstrates a simple example usage of the *TXLWizard* for generating TXL files with python code.

The code can be found in the file Content/Example_Simple.py. The resulting SVG image is shown in Figure *Generated SVG Image*.

A more advanced example is shown in Section *Advanced Example*

2.2.2 Code

```
1     ####################
2     # Import Libraries #
3     ####################
4
5     # Import TXLWriter, the main class for generating TXL Output
6     import TXLWizard.TXLWriter
7
8     # Import Pre-Defined Shapes / Structures wrapped in functions
9     import TXLWizard.ShapeLibrary.EndpointDetectionWindows
10    import TXLWizard.ShapeLibrary.Label
11
12    # Import math module for calculations
13    import math
14
15    ###############################
```
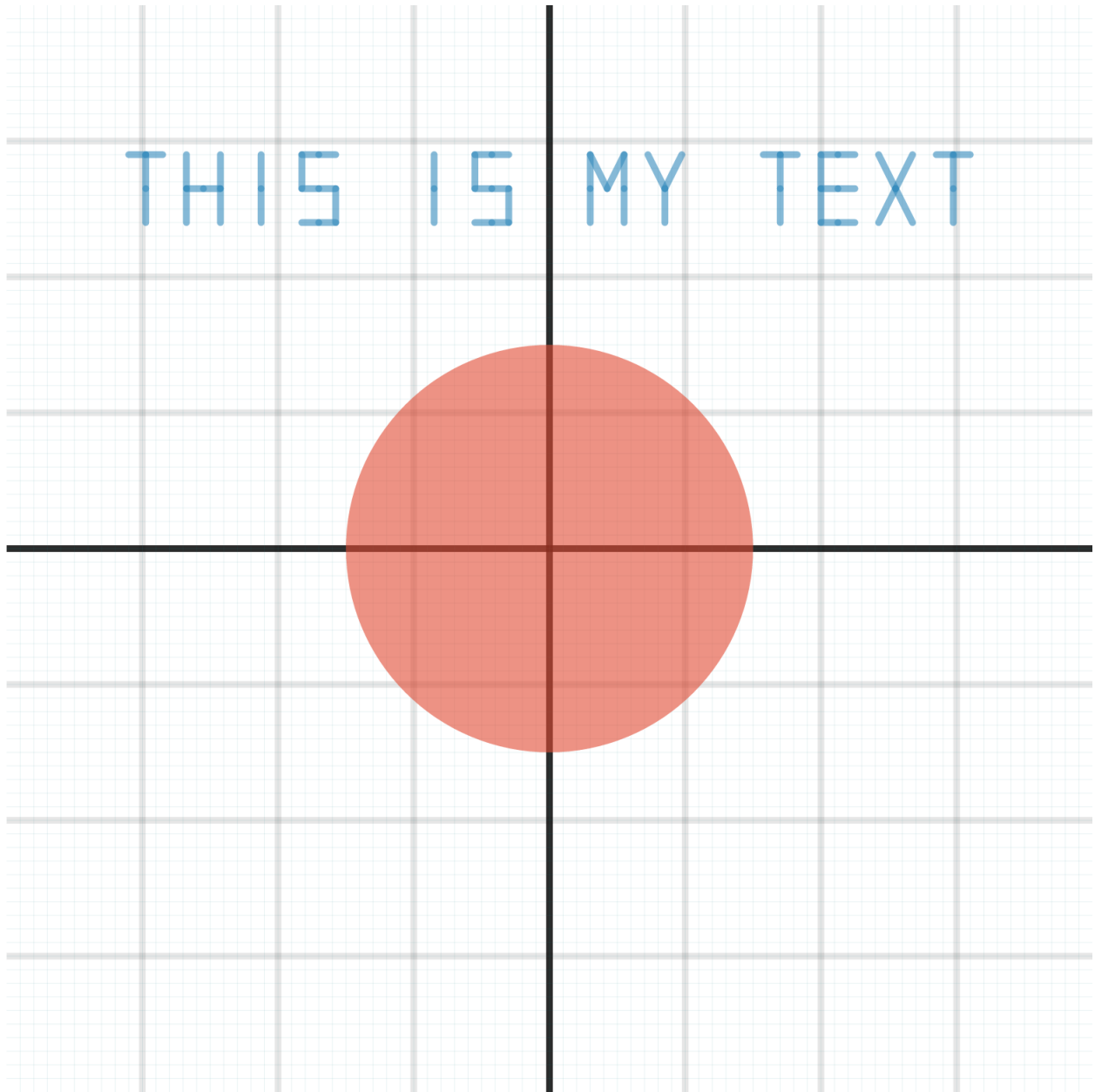
Fig. 2.1: Generated SVG Image for *Content/Example_Introduction.py*

```python
16  # Sample / Structure Parameters #
17  ################################
18
19  # Define all sample parameters
20  SampleParameters = {
21      'Width': 8e3,
22      'Height': 8e3,
23      'Label': 'Simple Demo',
24  }
25
26  # Define all structure parameters
27  StructureParameters = {
28      'Circle': {
29          'Radius': 50,
30          'Layer': 3
31      },
32      'CircleArray': {
33          'Columns': 6,
34          'Rows': 5,
35          'ArrayXOffset': 500,
36          'ArrayYOffset': -500,
37          'ArrayOrigin': [0.75e3, 3e3],
38          'Label': 'R{i}C{j}',
39      }
40  }
41
42  ########################
43  # Initialize TXLWriter #
44  ########################
45  TXLWriter = TXLWizard.TXLWriter.TXLWriter(
46      GridWidth=SampleParameters['Width'],
47      GridHeight=SampleParameters['Height']
48  )
49
50  ####################
51  # Define Structures #
52  ####################
53
54  ## Sample Label ##
55
56  # Give the sample a nice label
57  SampleLabelObject = TXLWizard.ShapeLibrary.Label.GetLabel(
58      TXLWriter,
59      Text=SampleParameters['Label'],
60      OriginPoint=[
61          0.5e3, 1. * SampleParameters['Height'] / 2. - 500
62      ],
63      FontSize=150,
64      StrokeWidth=20,
65      RoundCaps=True,  # Set to False to improve e-Beam performance
66      Layer=1
67  )
68
69  ## Endpoint Detection ##
70
71  # Use Pre-Defined Endpoint Detection Windows
72  TXLWizard.ShapeLibrary.EndpointDetectionWindows.GetEndpointDetectionWindows(
73      TXLWriter, Layer=1)
```

```
74
75   ## User Structure: Circle ##
76
77   # Create Definition Structure for Circle that will be reused
78   CircleStructure = TXLWriter.AddDefinitionStructure('MyCircleID')
79   CircleStructure.AddPattern(
80       'Circle',
81       Center=[0, 0],
82       Radius=StructureParameters['Circle']['Radius'],
83       Layer=StructureParameters['Circle']['Layer']
84
85   )
86
87   # Create array of the definition structure above
88   CircleArray = TXLWriter.AddContentStructure('MyCircleArray')
89   CircleArray.AddPattern(
90       'Array',
91       ReferencedStructureID=CircleStructure.ID,
92       OriginPoint=StructureParameters['CircleArray']['ArrayOrigin'],
93       PositionDelta1=[
94           StructureParameters['CircleArray']['ArrayXOffset'], 0
95       ],
96       PositionDelta2=[
97           0, StructureParameters['CircleArray']['ArrayYOffset']
98       ],
99       Repetitions1=StructureParameters['CircleArray']['Columns'],
100      Repetitions2=StructureParameters['CircleArray']['Rows']
101  )
102
103  ########################
104  # Generate Output Files #
105  ########################
106
107  # Note: The suffix (.txl, .html, .svg) will be appended automatically
108  TXLWriter.GenerateFiles('Masks/Example_Simple')
```

2.2.3 Generated SVG Image

2.3 TXLImport Example

2.3.1 Introduction

The following code demonstrates a simple example usage of the *TXLWizard* for importing existing TXL files and adding an array of labels.

The code can be found in the file Content/Example_ImportTXLFile.py. The resulting SVG image is shown in Figure *Generated SVG Image*.

A more advanced example is shown in Section *Advanced Example*
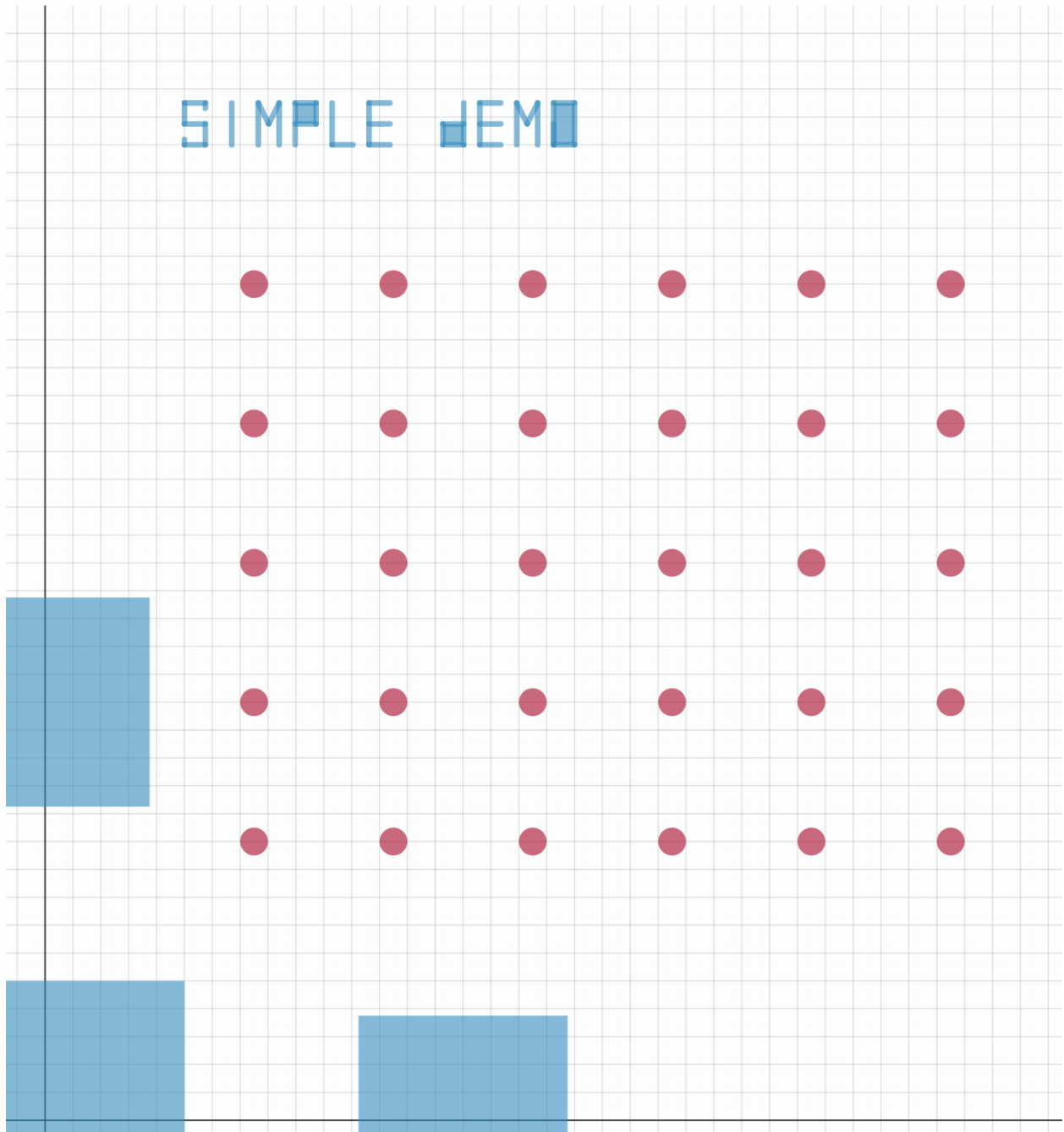
2.3.2 Code

```
1   ####################
2   # Import Libraries #
```

Fig. 2.2: Generated SVG Image for *Content/Example_Simple.py*

```python
3   ####################
4
5   # Import TXLWriter, the main class for generating TXL Output
6   import TXLWizard.TXLWriter
7
8   # Import Pre-Defined Shapes / Structures wrapped in functions
9   import TXLWizard.ShapeLibrary.LabelArray
10
11  ################################
12  # Sample / Structure Parameters #
13  ################################
14
15  # Define all sample parameters
16  SampleParameters = {
17      'Width': 8e3,
18      'Height': 8e3,
19      'Label': 'Simple Demo',
20  }
21
22  # Define all structure parameters
23  StructureParameters = {
24      'CircleArray': {
25          'Columns': 6,
26          'Rows': 5,
27          'ArrayXOffset': 500,
28          'ArrayYOffset': -500,
29          'ArrayOrigin': [0.75e3, 3e3],
30          'Label': 'R{j}C{i}',  # {i} and {j} will be replaced
31          # by str.format() with the corresponding auto-incremented index
32          'LabelXOffset': 0,
33          'LabelYOffset': 100,
34      }
35  }
36
37  ########################
38  # Initialize TXLWriter #
39  ########################
40  TXLWriter = TXLWizard.TXLWriter.TXLWriter(
41      GridWidth=SampleParameters['Width'],
42      GridHeight=SampleParameters['Height']
43  )
44
45  # Import existing TXL file
46  TXLWriter.ImportTXLFile('Masks/Example_Simple.txl')
47
48  # label each array element
49  TXLWizard.ShapeLibrary.LabelArray.GetLabelArray(
50      TXLWriter,
51      StructureParameters['CircleArray']['Label'],
52      OriginPoint=[
53          StructureParameters['CircleArray']['ArrayOrigin'][0]
54          + StructureParameters['CircleArray']['LabelXOffset'],
55          StructureParameters['CircleArray']['ArrayOrigin'][1]
56          + StructureParameters['CircleArray']['LabelYOffset']
57      ],
58      PositionDelta1=[
59          StructureParameters['CircleArray']['ArrayXOffset'], 0
60      ],
```

```
61      PositionDelta2=[
62          0, StructureParameters['CircleArray']['ArrayYOffset']
63      ],
64      Repetitions1=StructureParameters['CircleArray']['Columns'],
65      Repetitions2=StructureParameters['CircleArray']['Rows']
66  )
67
68  #########################
69  # Generate Output Files #
70  #########################
71
72  # Note: The suffix (.txl, .html, .svg) will be appended automatically
73  TXLWriter.GenerateFiles('Masks/Example_ImportTXLFile')
```
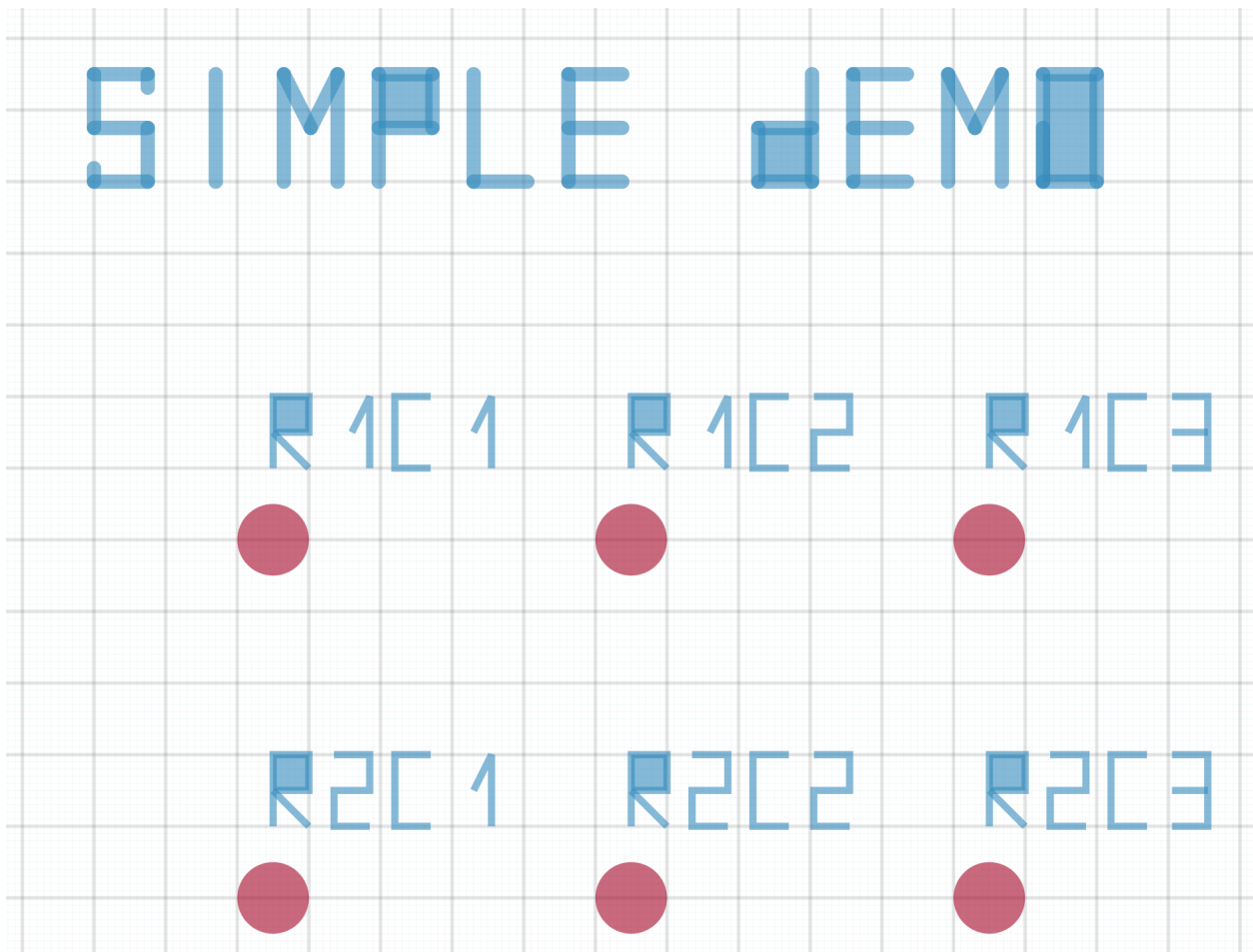
2.3.3 Generated SVG Image

Fig. 2.3: Generated SVG Image for *Content/Example_ImportTXLFile.py*

2.4 Advanced Example

2.4.1 Introduction

The following code demonstrates an advanced example usage of the *TXLWizard* for generating TXL files with python code.

The code can be found in the file Content/Example_Advanced.py. The resulting SVG image is shown in Figure *Generated SVG Image*.

2.4.2 Code

```python
1   ####################
2   # Import Libraries #
3   ####################
4
5   # Import TXLWriter, the main class for generating TXL Output
6   import TXLWizard.TXLWriter
7
8   # Import Pre-Defined Shapes / Structures wrapped in functions
9   import TXLWizard.ShapeLibrary.EndpointDetectionWindows
10  import TXLWizard.ShapeLibrary.Label
11  import TXLWizard.ShapeLibrary.LabelArray
12  import TXLWizard.ShapeLibrary.AlignmentMarkers
13  import TXLWizard.ShapeLibrary.CornerCube
14
15  # Import math module for calculations
16  import math
17
18
19  ################################
20  # Sample / Structure Parameters #
21  ################################
22
23  # Define all sample parameters
24  SampleParameters = {
25      'Width': 8e3,
26      'Height': 8e3,
27      'Label': 'GOI Demo CornerCube',
28  }
29
30  # Define all structure parameters
31  StructureParameters = {
32      'CornerCube': {
33          'BridgeLength':40,
34          'ParabolaFocus': 45,
35          'XCutoff': 45,
36          'AirGapX': 15,
37          'AirGapY': 5,
38          'LabelXOffset': 0,
39          'LabelYOffset': 150,
40          'Label': 'R{i}C{j}', # {i} and {j} will be replaced
41                              # by str.format() with the corresponding row / column index
42          'Layer': 2
43      },
44      'Circle': {
```

```
45          'Radius': 25,
46          'Layer': 3
47      },
48      'CornerCubeArray': {
49          'Columns': 6,
50          'Rows': 3,
51          'ArrayXOffset': 500,
52          'ArrayYOffset': -500,
53          'ArrayOrigin': [0.75e3, 3e3]
54      }
55  }
56
57
58  ########################
59  # Initialize TXLWriter #
60  ########################
61  TXLWriter = TXLWizard.TXLWriter.TXLWriter(
62      GridWidth=SampleParameters['Width'],
63      GridHeight=SampleParameters['Height'],
64      Precision=6 #increase the precision / resolution to 0.000001 (10^-6)
65  )
66
67  #####################
68  # Define Structures #
69  #####################
70
71  ## Sample Label ##
72
73  # Give the sample a nice label...
74  SampleLabelObject = TXLWizard.ShapeLibrary.Label.GetLabel(
75      TXLWriter,
76      Text=SampleParameters['Label'],
77      OriginPoint=[
78          0.5e3, 1. * SampleParameters['Height'] / 2. - 500
79      ],
80      FontSize=150,
81      StrokeWidth=20,
82      RoundCaps=False,# Set to False to improve e-Beam performance
83      Layer=1
84  )
85  # ...and some other information
86  Alphabet = TXLWizard.ShapeLibrary.Label.GetLabel(
87      TXLWriter,
88      Text='abcdefghijklmnopqrstuvwxyz0123456789 megamega ggg ah extraaaa rischaaaar',
89      OriginPoint=[
90          0.5e3, 1. * SampleParameters['Height'] / 2. - 600
91      ],
92      FontSize=50,
93      StrokeWidth=3,
94      RoundCaps=False, # Set to False to improve e-Beam performance
95      Layer=1
96  )
97
98  ## Endpoint Detection ##
99
100 # Use Pre-Defined Endpoint Detection Windows
101 TXLWizard.ShapeLibrary.EndpointDetectionWindows.GetEndpointDetectionWindows(
102     TXLWriter, Layer=1
```

```
103  )
104
105  ## Alignment Markers ##
106
107  # Use Pre-Defined Alignment Markers
108  TXLWizard.ShapeLibrary.AlignmentMarkers.GetAlignmentMarkers(
109      TXLWriter, Layer=1
110  )
111
112
113  ## User Structure: Corner Cube ##
114
115  # Create Definition Structure for Corner Cube that will be reused
116  CornerCubeDefinition = TXLWizard.ShapeLibrary.CornerCube.GetCornerCube(
117      TXLWriter,
118      ParabolaFocus=StructureParameters['CornerCube']['ParabolaFocus'],
119      XCutoff=StructureParameters['CornerCube']['XCutoff'],
120      AirGapX=StructureParameters['CornerCube']['AirGapX'],
121      AirGapY=StructureParameters['CornerCube']['AirGapY'],
122      Layer=StructureParameters['CornerCube']['Layer']
123  )
124
125  # Create Definition Structure for combination of cornercube and additional circle
126  FullCornerCubeNoRotation = TXLWriter.AddDefinitionStructure('FullCornerCubeNoRotation')
127  FullCornerCubeNoRotation.AddPattern(
128      'Reference',
129      ReferencedStructureID=CornerCubeDefinition.ID,
130      OriginPoint=[1. * StructureParameters['CornerCube']['BridgeLength'] / 2., 0]
131  )
132  FullCornerCubeNoRotation.AddPattern(
133      'Circle',
134      Center=[0, 0],
135      Radius=StructureParameters['Circle']['Radius'],
136      Layer=StructureParameters['Circle']['Layer']
137  )
138
139  # Create definition structure with rotation of entire referenced structure
140  FullCornerCube = TXLWriter.AddDefinitionStructure('FullCornerCube',
141                                                    RotationAngle=45)
142  FullCornerCube.AddPattern(
143      'Reference',
144      ReferencedStructureID=FullCornerCubeNoRotation.ID,
145      OriginPoint=[0, 0]
146  )
147
148  # Create array of the definition structure above
149  CornerCubeArrayFine = TXLWriter.AddContentStructure('CornerCubeArrayFine')
150  CornerCubeArrayFine.AddPattern(
151      'Array',
152      ReferencedStructureID=FullCornerCube.ID,
153      OriginPoint=StructureParameters['CornerCubeArray']['ArrayOrigin'],
154      PositionDelta1=[
155          StructureParameters['CornerCubeArray']['ArrayXOffset'], 0
156      ],
157      PositionDelta2=[
158          0, StructureParameters['CornerCubeArray']['ArrayYOffset']
159      ],
160      Repetitions1=StructureParameters['CornerCubeArray']['Columns'],
```

```
161        Repetitions2=StructureParameters['CornerCubeArray']['Rows']
162    )
163
164    # Add a label array to label each element of the array pattern above
165    TXLWizard.ShapeLibrary.LabelArray.GetLabelArray(
166        TXLWriter,
167        StructureParameters['CornerCube']['Label'],
168        OriginPoint=[
169            StructureParameters['CornerCubeArray']['ArrayOrigin'][0]
170            + StructureParameters['CornerCube']['LabelXOffset'],
171            StructureParameters['CornerCubeArray']['ArrayOrigin'][1]
172            + StructureParameters['CornerCube']['LabelYOffset']
173        ],
174        PositionDelta1=[
175            StructureParameters['CornerCubeArray']['ArrayXOffset'], 0
176        ],
177        PositionDelta2=[
178            0, StructureParameters['CornerCubeArray']['ArrayYOffset']
179        ],
180        Repetitions1=StructureParameters['CornerCubeArray']['Columns'],
181        Repetitions2=StructureParameters['CornerCubeArray']['Rows'],
182        FontSize=40,
183        StrokeWidth=3,
184        RoundCaps=False,# Set to False to improve e-Beam performance
185        Layer=1,
186        RotationAngle=45
187    )
188
189
190
191    #########################
192    # Generate Output Files #
193    #########################
194
195    # Note: The suffix (.txl, .html, .svg) will be appended automatically
196    TXLWriter.GenerateFiles('Masks/Example_Advanced')
197
```
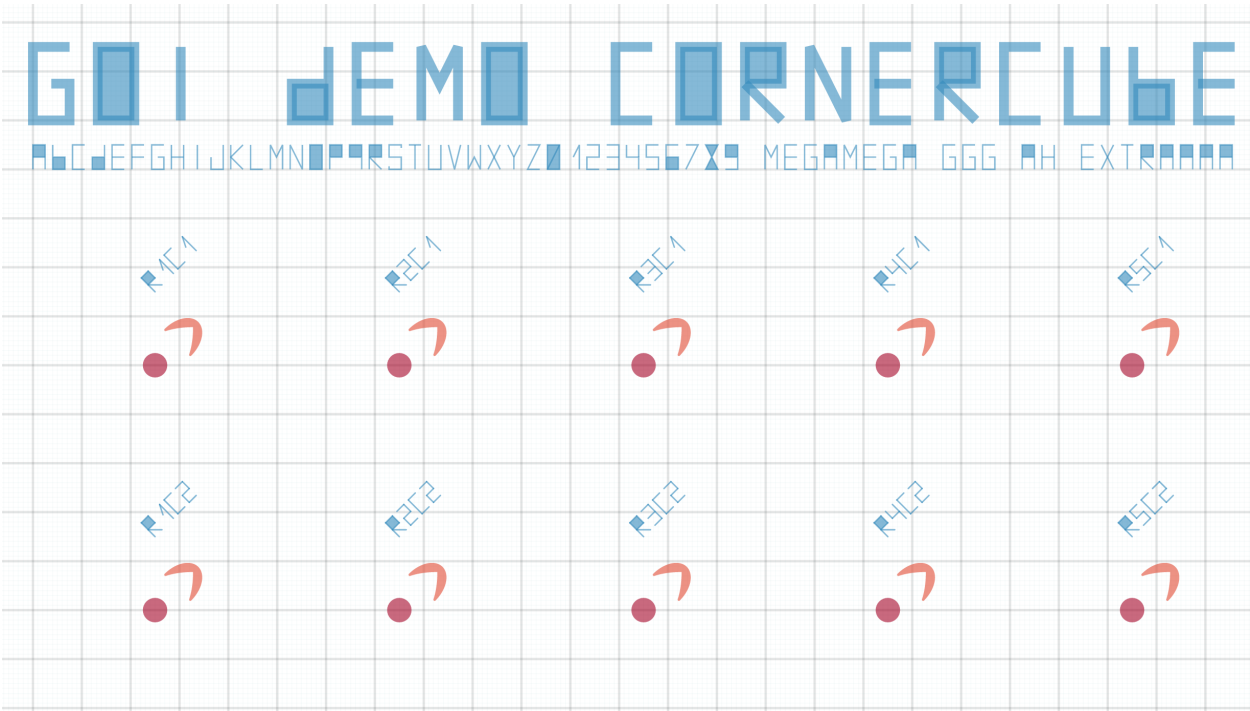
2.4.3 Generated SVG Image

Fig. 2.4: Generated SVG Image for *Content/Example_Advanced.py*

TXLCONVERTER

For existing TXL files, there is a command line interface script that converts them to SVG / HTML files.

3.1 Usage

The usage is very simple. Simply run the python script *TXLWizard/TXLConverterCLI.py*. The command line interface will allow you to change the configuration as you wish. Furthermore, the configuration is saved and restored for a subsequent run.

3.1.1 Code

To use the *TXLConverter* from the command line type

```
python TXLWizard/Tools/TXLConverterCLI.py
```

Or if you want to call it in your own python script do

```python
import TXLWizard.TXLConverter
TXLConverterCLI = TXLWizard.TXLConverter.TXLConverterCLI()
```

The resulting command line interface looks as follows:

```
### TXL Converter v1.6 ###
Converts TXL Files to SVG/HTML
written by Esteban Marin (estebanmarin@gmx.ch)


Full TXL File / Folder Path
If the path is a folder, you can enter the filename separately.
[/home/john.mega/masks]:
/Users/esteban/Desktop/masks2/tmpd/EM160225_GOI_CornerCube_Microbridge.txl

SampleWidth in um
used to draw coordinate system
[1500]:

SampleHeight in um
used to draw coordinate system
[1500]:

Layers to process
comma-separated, e.g. 1,4,5. Type -1 for all layers.
[-1]:
```

```
Do Conversion (y/n)? [y]

Files written:
/Users/esteban/Desktop/masks2/tmpd/EM160225_GOI_CornerCube_Microbridge.html
/Users/esteban/Desktop/masks2/tmpd/EM160225_GOI_CornerCube_Microbridge.svg

Done
```

PYTHON MODULE REFERENCE

4.1 TXLWriter

TXLWizard.TXLWriter	Controller class for generating TXL / SVG / HTML output.

4.1.1 TXLWizard.TXLWriter

Controller class for generating TXL / SVG / HTML output.

Here we can add structures (definitions and content) which will be rendered in the output.

Classes

TXLWriter(**kwargs)	Controller class for generating TXL / SVG / HTML output.

class TXLWizard.TXLWriter.**TXLWriter**(***kwargs*)
    Bases: object

    Controller class for generating TXL / SVG / HTML output.

    Here we can add structures (definitions and content) which will be rendered in the output.

    Optionally, a coordinate system grid is drawn.

        Parameters

- **ShowGrid** (*bool, optional*) – Show the coordinate system grid or not.

  Defaults to True

- **GridWidth** (*int, optional*) – Full width of the coordinate system grid in um.

  Defaults to 800

- **GridHeight** (*int, optional*) – Full height of the coordinate system grid in um.

  Defaults to 800

- **GridSpacing** (*int, optional*) – Coordinate Sytem Grid Spacing in um.

  Defaults to 100

- **SubGridSpacing** (*int, optional*) – Coordinate System Sub-Grid Spacing in um.

  Defaults to 10

- **Precision** (*int, optional*) – Number of digits for float to str conversion / Resolution of TXL file.

  Defaults to 4

Examples

Import required Modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter(
...     ShowGrid=True, GridWidth=800, GridHeight=800
... )
```

Add a definition structure and add a pattern of type *Circle*

```
>>> MyDefinitionStructure = TXLWriter.AddDefinitionStructure('MyDefinition')
>>> MyDefinitionStructure.AddPattern('Circle', Center=[0,0], Radius=20, Layer=3)
<TXLWizard.Patterns.Circle.Circle object at 0x...>
```

Add a content structure with a pattern *Reference* to reuse the definition structure.

```
>>> MyContentStructure = TXLWriter.AddContentStructure('MySuperCircle')
>>> MyContentStructure.AddPattern(
...     'Reference',
...     ReferencedStructureID=MyDefinitionStructure.ID,
...     OriginPoint=[20,50]
... )
<TXLWizard.Patterns.Reference.Reference object at 0x...>
```

Generate the Output files with name *Example_TXLWriter.(txl|html|svg)* to the folder *Tests/Results*

```
>>> TXLWriter.GenerateFiles('Tests/Results/TXLWriter/Example_TXLWriter')
```

AddContentStructure(*ID*, ***kwargs*)

Add content structure. A content structure can hold patterns that will render in the output.

A structure corresponds to the "STRUCT" command in the TXL file format.

> **Parameters**
>
> - **ID** (*str*) – Unique identification of the structure. Must be used when referencing to this structure.
> - **kwargs** (*dict*) – Keyword arguments passed to the structure constructor. See *TXLWizard.Patterns.Structure.Structure*
>
> **Returns**
>
> **Return type** *TXLWizard.Patterns.Structure.Structure* structure instance

AddDefinitionStructure(*ID*, ***kwargs*)

Add definition structure. A definition structure can be referenced by a content structure.

A structure corresponds to the "STRUCT" command in the TXL file format.

> **Parameters**
>
> - **ID** (*str*) – Unique identification of the structure. Must be used when referencing to this structure.

- **kwargs** (*dict*) – Keyword arguments passed to the structure constructor. See *TXLWizard.Patterns.Structure.Structure*

  Returns

  Return type *TXLWizard.Patterns.Structure.Structure* structure instance

GenerateFiles(*Filename*, *TXL=True*, *SVG=True*, *HTML=True*, *TargetFolder=None*)
  Generate the output files (.txl, .svg, .html).

  Parameters

  - **Filename** (*str*) – Path / Filename without extension. The corresponding path will be created if it does not exist

  - **TXL** (*bool, optional*) – Enable TXL Output.

    Defaults to True

  - **SVG** (*bool, optional*) – Enable SVG Output.

    Defaults to True

  - **HTML** (*bool, optional*) – Enable HTML Output. If set to *True*, also *SVG* needs to be set to *True*

    Defaults to True

  - **TargetFolder** (*str, optional*) – If given, the generated files are stored in the folder specified.

    If not given, the generated files are stored in the path specified in *Filename*

    Defaults to None.

ImportTXLFile(*Filename*, *LayersToProcess=[]*)
  Import an existing TXL file for further processing.

  The content structures can be accessed with *self._ContentStructures* (read-only!).

  The order of the content structures is stored in *self._ContentStructuresIndexList* (read-only!).

  The definition structures are stored in *self._Definitions.Structures*

  Parameters

  - **Filename** (*str*) – Path / Filename of the .txl file to be imported

  - **LayersToProcess** (*list of int, optional*) – if given, only layers in this list are processed / shown.

    Defaults to []

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
>>> import TXLWizard.ShapeLibrary.Label
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

import TXL file *myPath/mask_orig.txl*

```
>>> TXLWriter.ImportTXLFile('Tests/SampleFiles/Example_Simple_Original.txl', LayersToProcess
```

Get an existing structure and add a pattern to it

```
>>> MyStructure = TXLWriter._ContentStructures['MyCircleArray']
>>> MyStructure.AddPattern(
...     'Circle',
...     Center=[-100, 0],
...     Radius=50,
...     Layer=1
... )
<TXLWizard.Patterns.Circle.Circle object at 0x...>
```

Add a label

```
>>> SampleLabelObject = TXLWizard.ShapeLibrary.Label.GetLabel(
...     TXLWriter,
...     Text='This is my text',
...     OriginPoint=[
...         -200, 300
...     ],
...     FontSize=150,
...     StrokeWidth=20
... )
```

Generate the Output files with name *ExampleSimple_Modified.(txl|html|svg)* to the folder *Tests/Results*

```
>>> TXLWriter.GenerateFiles('Tests/Results/TXLWriter/Example_Simple_Modified')
```

4.2 Patterns

TXLWizard.Patterns.AbstractPattern	Provides an abstract class for *Pattern* objects
TXLWizard.Patterns.Array	Implements a class for *Pattern* objects of type *Array* (*AREF*).
TXLWizard.Patterns.Circle	Implements a class for *Pattern* objects of type *Circle* (*C*).
TXLWizard.Patterns.Ellipse	Implements a class for *Pattern* objects of type *Ellipse* (*ELP*).
TXLWizard.Patterns.Polygon	Implements a class for *Pattern* objects of type *Polygon* (*B*).
TXLWizard.Patterns.Polyline	Implements a class for *Pattern* objects of type *Polyline* (*B*).
TXLWizard.Patterns.Reference	Implements a class for *Pattern* objects of type *Reference* (*SREF*).
TXLWizard.Patterns.Structure	Implements a class for *Structure* objects (*STRUCT*).

4.2.1 TXLWizard.Patterns.AbstractPattern

Provides an abstract class for *Pattern* objects

Classes

AbstractPattern(**kwargs)	Provides an abstract class for *Pattern* objects.

class TXLWizard.Patterns.AbstractPattern.**AbstractPattern**(**kwargs*)

Bases: object

Provides an abstract class for *Pattern* objects.

> **Parameters**
>
> - **Layer** (`int, optional`) – Specifies the *Layer* attribute of the pattern.
>
>   Defaults to None.
>
> - **DataType** (`int, optional`) – Specifies the *DataType* attribute of the pattern.
>
>   Defaults to None.
>
> - **RotationAngle** (`float, optional`) – Specifies the *RotationAngle* attribute of the pattern.
>
>   Defaults to None.
>
> - **StrokeWidth** (`float, optional`) – Specifies the *StrokeWidth* attribute of the pattern.
>
>   Defaults to None.
>
> - **ScaleFactor** (`float, optional`) – Specifies the *ScaleFactor* attribute of the pattern.
>
>   Defaults to None.

Attributes = None
> *dict* – attribute values of the current pattern. Default values are copied from *self.DefaultAttributes*

DefaultAttributes = None
> *dict* – default attributes that are copied to *self.Attributes* upon instantiation. Specifies the allowed attributes

GetSVGOutput ()
> Generates the SVG output xml for the current pattern. Needs to be implemented for each pattern type separately in the corresponding inheriting class.
>
> > **Returns** SVG output xml
> >
> > **Return type** str

GetTXLOutput ()
> Generates the TXL output commands for the current pattern. Needs to be implemented for each pattern type separately in the corresponding inheriting class.
>
> > **Returns** TXL output commands
> >
> > **Return type** str

ParentStructure = None
> *TXLWizard.Patterns.Structure.Structure*, reference to the *Structure* instance containing the current pattern

Type = None
> *str* – specifies the type of the pattern.

4.2.2 TXLWizard.Patterns.Array

Implements a class for *Pattern* objects of type *Array* (*AREF*).

Replicates the referenced structure in two directions.

Classes

Array(ReferencedStructureID, OriginPoint, ...)	Implements a class for *Pattern* objects of type *Array*.

class TXLWizard.Patterns.Array.**Array**(*ReferencedStructureID*, *OriginPoint*, *PositionDelta1*, *PositionDelta2*, *Repetitions1*, *Repetitions2*, ***kwargs*)

Bases: *TXLWizard.Patterns.AbstractPattern.AbstractPattern*

Implements a class for *Pattern* objects of type *Array*.

Corresponds to the TXL command *AREF*.

Replicates the referenced structure *ReferencedStructureID* in two directions *PositionDelta1* and *PositionDelta2* for the number of times specified in *Repetitions1* and *Repetitions2*, starting at *OriginPoint*.

The x- and y-coordinates of the replicated objects are calculated as follows: *OriginPoint+i*PositionDelta1+j*PositionDelta2* where *i* is an integer that ranges from 0 to *Repetitions1 - 1* and *j* is an integer that ranges from 0 to *Repetitions2 - 1*

> **Parameters**
>
> - **ReferencedStructureID** (*str*) – ID of the structure being referenced to
> - **OriginPoint** (*list of float*) – x- and y- coordinates of the starting point
> - **PositionDelta1** (*list of float*) – x- and y- coordinates of the first replication direction.
> - **PositionDelta2** (*list of float*) – x- and y- coordinates of the second replication direction.
> - **Repetitions1** (*int*) – Number of replications in the first replication direction
> - **Repetitions2** (*int*) – Number of replications in the second replication direction
> - ****kwargs** – keyword arguments passed to the *TXLWizard.Patterns.AbstractPattern.AbstractPa* constructor. Can specify attributes of the current pattern.

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Create Definition Structure for Circle that will be reused. Could also be a content structure.

```
>>> CircleStructure = TXLWriter.AddDefinitionStructure('MyCircleID')
>>> CircleStructure.AddPattern(
...     'Circle',
...     Center=[0, 0],
...     Radius=50,
...     Layer=1
... )
<TXLWizard.Patterns.Circle.Circle object at 0x...>
```

Create array of the definition structure above with 10 repetitions at distance 100 in x-direction 20 repetitions at distance 200 in y-direction

```
>>> CircleArray = TXLWriter.AddContentStructure('MyCircleArray')
>>> CircleArray.AddPattern(
...     'Array',
...     ReferencedStructureID=CircleStructure.ID,
...     OriginPoint=[40,60],
...     PositionDelta1=[
...         100, 0
...     ],
...     PositionDelta2=[
...         0, 200
...     ],
...     Repetitions1=10,
...     Repetitions2=20
... )
<TXLWizard.Patterns.Array.Array object at 0x...>
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/Patterns/Array')
```

PositionDelta1 = None
> *list of float* – x- and y- coordinates of the first replication direction.

PositionDelta2 = None
> *list of float* – x- and y- coordinates of the second replication direction.

ReferencedStructureID = None
> *str* – ID of the structure being referenced to

Repetitions1 = None
> *int* – Number of replications in the first replication direction

Repetitions2 = None
> *int* – Number of replications in the second replication direction

Type = None
> *str* – specifies the type of the pattern. Set to 'Array'

4.2.3 TXLWizard.Patterns.Circle

Implements a class for *Pattern* objects of type *Circle* (*C*).

Renders a circle.

Classes

[*Circle*](Center, Radius, **kwargs)	Implements a class for *Pattern* objects of type *Circle*.

class TXLWizard.Patterns.Circle.**Circle**(*Center*, *Radius*, ***kwargs*)
> Bases: *TXLWizard.Patterns.AbstractPattern.AbstractPattern*

> Implements a class for *Pattern* objects of type *Circle*.

> Corresponds to the TXL command *C* (*CP* if *PathOnly* is specified, *CPR* if *RoundCaps* and *CPE* if *Extended*).

> Renders a circle.

> Optionally, only a sector is shown when specifying *StartAngle* and *EndAngle*.

If *NumberOfPoints* is given, the number of path segments defining the circle can be specified.

If *PathOnly* is set to True, only the arc of the circle is shown. Optionally, the ends of the path are rounded by specifying *RoundCaps* or extended by specifying *Extended* along with *PathOnly*.

> **Parameters**
>
> - **Center** (`list of float`) – x- and y-coordinates specifying the center of the circle
>
> - **Radius** (`float`) – Radius of the circle
>
> - **StartAngle** (`float, optional`) – If given, only a sector is drawn from *StartAngle* to *EndAngle*.
>
>   Defaults to None.
>
> - **EndAngle** (`float, optional`) – If given, only a sector is drawn from *StartAngle* to *EndAngle*.
>
>   Defaults to None.
>
> - **NumberOfPoints** (`int, optional`) – Number of path segments used for drawing the circle.
>
>   Defaults to None.
>
> - **PathOnly** (`bool, optional`) – If set to True, only the arc of the circle is drawn.
>
>   Defaults to False.
>
> - **RoundCaps** (`bool, optional`) – If set to True along with *PathOnly*, the end of the path is rounded.
>
>   Defaults to False.
>
> - **Extended** (`bool, optional`) – If set to True along with *PathOnly*, the end of the path is extended.
>
>   Defaults to False.
>
> - ****kwargs** – keyword arguments passed to the *TXLWizard.Patterns.AbstractPattern.AbstractPa* constructor. Can specify attributes of the current pattern.

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Create Content Structure for Circle and add Pattern of type *Circle*

```
>>> CircleStructure = TXLWriter.AddContentStructure('MyCircleID')
>>> CircleStructure.AddPattern(
...     'Circle',
...     Center=[0, 0],
...     Radius=50,
...     Layer=1
... )
<TXLWizard.Patterns.Circle.Circle object at 0x...>
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/Patterns/Circle')
```

Center = None
> *list of float* – x- and y-coordinates specifying the center of the circle

EndAngle = None
> *float* – If set, only a sector is drawn from *self.StartAngle* to *self.EndAngle*.

EndPoint = None
> *list of float* – If *self.StartAngle* and *self.EndAngle* are set, the ending point of the segment arc is calculated

Extended = None
> *bool* – If set to True along with *PathOnly*, the end of the path is extended

NumberOfPoints = None
> *int* – Number of path segments used for drawing the circle.

PathOnly = None
> *bool* – If set to True, only the arc of the circle is drawn.

Radius = None
> *float* – Radius of the circle

RoundCaps = None
> *bool* – If set to True along with *PathOnly*, the end of the path is rounded

StartAngle = None
> *float* – If set, only a sector is drawn from *self.StartAngle* to *self.EndAngle*.

StartPoint = None
> *list of float* – If *self.StartAngle* and *self.EndAngle* are set, the starting point of the segment arc is calculated

Type = None
> *str* – specifies the type of the pattern. Set to 'Circle'

4.2.4 TXLWizard.Patterns.Ellipse

Implements a class for *Pattern* objects of type *Ellipse* (*ELP*).

Renders an ellipse.

Classes

Ellipse(Center, RadiusX, RadiusY, **kwargs)	Implements a class for *Pattern* objects of type *Ellipse*.

class TXLWizard.Patterns.Ellipse.**Ellipse**(*Center*, *RadiusX*, *RadiusY*, ***kwargs*)
> Bases: *TXLWizard.Patterns.AbstractPattern.AbstractPattern*

> Implements a class for *Pattern* objects of type *Ellipse*.

> Corresponds to the TXL command *ELP*.

> Renders an ellipse. Optionally, only a sector is shown when specifying *StartAngle* and *EndAngle*.

> If *NumberOfPoints* is given, the number of path segments defining the ellipse can be specified.

> If *PathOnly* is set to True, only the arc of the ellipse is shown.

Parameters

- **Center** (*list of float*) – x- and y-coordinates specifying the center of the ellipse
- **RadiusX** (*float*) – Semi-major axis of the ellipse in x-direction
- **RadiusY** (*float*) – Semi-minor axis of the ellipse in y-direction
- **StartAngle** (*float, optional*) – If given, only a sector is drawn from *StartAngle* to *EndAngle*.

  Defaults to 0

- **EndAngle** (*float, optional*) – If given, only a sector is drawn from *StartAngle* to *EndAngle*.

  Defaults to 0

- **NumberOfPoints** (*int, optional*) – Number of path segments used for drawing the ellipse.

  Defaults to None.

- ****kwargs** – keyword arguments passed to the *TXLWizard.Patterns.AbstractPattern.AbstractPa* constructor. Can specify attributes of the current pattern.

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Create Content Structure for ellipse and add Pattern of type *Ellipse*

```
>>> EllipseStructure = TXLWriter.AddContentStructure('MyEllipseID')
>>> EllipseStructure.AddPattern(
...     'Ellipse',
...     Center=[0, 0],
...     RadiusX=50,
...     RadiusY=70,
...     Layer=1
... )
<TXLWizard.Patterns.Ellipse.Ellipse object at 0x...>
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/Patterns/Ellipse')
```

Center = None
   list of float – x- and y-coordinates specifying the center of the ellipse

EndAngle = None
   float – If given, only a sector is drawn from *StartAngle* to *EndAngle*.

EndPoint = None
   list of float – If *self.StartAngle* and *self.EndAngle* are set, the ending point of the segment arc is calculated

NumberOfPoints = None
   int – Number of path segments used for drawing the ellipse.

RadiusX = None
> *float* – Semi-major axis of the ellipse in x-direction

RadiusY = None
> *float* – Semi-minor axis of the ellipse in y-direction

StartAngle = None
> *float* – If given, only a sector is drawn from *StartAngle* to *EndAngle*.

StartPoint = None
> *list of float* – If *self.StartAngle* and *self.EndAngle* are set, the starting point of the segment arc is calculated

Type = None
> *str* – specifies the type of the pattern. Set to 'Ellipse'

4.2.5 TXLWizard.Patterns.Polygon

Implements a class for *Pattern* objects of type *Polygon* (*B*).

Renders an polygon.

Classes

Polygon(Points, **kwargs)	Implements a class for *Pattern* objects of type *Polygon*.

class TXLWizard.Patterns.Polygon.**Polygon**(*Points*, ***kwargs*)
> Bases: *TXLWizard.Patterns.AbstractPattern.AbstractPattern*

> Implements a class for *Pattern* objects of type *Polygon*.

> Corresponds to the TXL command *B*

> Renders an polygon.

> The boundary is always closed so the last point connects to the starting point

> > **Parameters**
> >
> > - **Points** (*list of list of float*) – List of points (each point is a list of float, specifying the x- and y-coordinate of the point) that define the polygon
> >
> > - ****kwargs** – keyword arguments passed to the *TXLWizard.Patterns.AbstractPattern.AbstractPa* constructor. Can specify attributes of the current pattern.

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Create Content Structure for polygon and add Pattern of type *Polygon*

```
>>> PolygonStructure = TXLWriter.AddContentStructure('MyPolygonID')
>>> PolygonStructure.AddPattern(
...     'Polygon',
...     Points=[[0,0], [0,10], [20,50], [0,0]],
...     Layer=1
... )
<TXLWizard.Patterns.Polygon.Polygon object at 0x...>
```

Complex structures can easily be added by generating the polygon points

```
>>> import math
>>> PolygonPoints = []
>>> Radius = 5.
>>> for i in range(21):
...     # AngleRadians goes from 0 to pi in 20 steps
...     AngleRadians = 0.5*2.*math.pi*1./20.*i
...     PolygonPoints.append([
...         Radius*math.cos(AngleRadians),Radius*math.sin(AngleRadians)
...     ])
>>> PolygonPoints.append([-20,-30])
>>> PolygonPoints.append([20,-30])
>>>
>>> PolygonStructure.AddPattern(
...     'Polygon',
...     Points=PolygonPoints,
...     Layer=1
... )
<TXLWizard.Patterns.Polygon.Polygon object at 0x...>
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/Patterns/Polygon')
```

Points = None
> *list of list of float* – List of points (each point is a list of float, specifying the x- and y-coordinate of the point) that define the polygon

Type = None
> *str* – specifies the type of the pattern. Set to 'Polygon'

4.2.6 TXLWizard.Patterns.Polyline

Implements a class for *Pattern* objects of type *Polyline* (*B*).

Renders an path specified by points.

Classes

Polyline(Points, **kwargs)	Implements a class for *Pattern* objects of type *Polyline*.

class TXLWizard.Patterns.Polyline.**Polyline**(*Points*, ***kwargs*)
> Bases: *TXLWizard.Patterns.AbstractPattern.AbstractPattern*

> Implements a class for *Pattern* objects of type *Polyline*.

> Corresponds to the TXL command *P* (*PR* if *RoundCaps* is True).

Renders an path specified by points.

The ends can be rounded by specifying *RoundCaps*

> **Parameters**
>
> > • **Points** (*list of list of float*) – List of points (each point is a list of float, specifying the x- and y-coordinate of the point) that define the path
> >
> > • **RoundCaps** (*bool, optional*) – If set to True, the end of the path is rounded.
> >
> > Defaults to False.
> >
> > • ****kwargs** – keyword arguments passed to the *TXLWizard.Patterns.AbstractPattern.AbstractPa* constructor. Can specify attributes of the current pattern.

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Create Content Structure for polyline and add Pattern of type *Polyline*

```
>>> PolylineStructure = TXLWriter.AddContentStructure('MyPolylineID')
>>> PolylineStructure.AddPattern(
...     'Polyline',
...     Points=[[0,0], [0,10], [20,50], [0,0]],
...     StrokeWidth=3,
...     Layer=1
... )
<TXLWizard.Patterns.Polyline.Polyline object at 0x...>
```

Complex structures can easily be added by generating the Polyline points

```
>>> import math
>>> PolylinePoints = []
>>> Radius = 10.
>>> for i in range(21):
...     # AngleRadians goes from 0 to pi in 20 steps
...     AngleRadians = 0.5*2.*math.pi*1./20.*i
...     PolylinePoints.append([
...         Radius*math.cos(AngleRadians),Radius*math.sin(AngleRadians)
...     ])
>>> PolylinePoints.append([-20,-30])
>>> PolylinePoints.append([20,-30])
>>>
>>> PolylineStructure.AddPattern(
...     'Polyline',
...     Points=PolylinePoints,
...     RoundCaps=True,
...     StrokeWidth=3,
...     Layer=1
... )
<TXLWizard.Patterns.Polyline.Polyline object at 0x...>
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/Patterns/Polyline')
```

Points = None
> *list of list of float* – List of points (each point is a list of float, specifying the x- and y-coordinate of the point) that define the polygon

RoundCaps = None
> *bool* – If set to True, the end of the path is rounded

Type = None
> *str* – specifies the type of the pattern. Set to 'Polyline'

4.2.7 TXLWizard.Patterns.Reference

Implements a class for *Pattern* objects of type *Reference* (*SREF*).

Renders a copy of the referenced structure.

Classes

Reference(ReferencedStructureID, ...)	Implements a class for *Pattern* objects of type *Reference*.

class TXLWizard.Patterns.Reference.**Reference**(*ReferencedStructureID*, *OriginPoint*, ***kwargs*)
> Bases: *TXLWizard.Patterns.AbstractPattern.AbstractPattern*

Implements a class for *Pattern* objects of type *Reference*.

Corresponds to the TXL command *SREF*.

Renders a copy of the structure identified by *ReferencedStructureID* at *OriginPoint*.

> **Parameters**
>
> - **ReferencedStructureID** (`str`) – ID of the structure being referenced to
>
> - **OriginPoint** (`list of float`) – x- and y-coordinates of the starting point
>
> - ****kwargs** – keyword arguments passed to the *TXLWizard.Patterns.AbstractPattern.AbstractPa* constructor. Can specify attributes of the current pattern.

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Create Content Structure for Circle that will be reused. Could also be a definition structure.

```
>>> CircleStructure = TXLWriter.AddContentStructure('MyCircleID')
>>> CircleStructure.AddPattern(
...     'Circle',
...     Center=[0, 0],
```

```
...        Radius=50,
...        Layer=1
... )
<TXLWizard.Patterns.Circle.Circle object at 0x...>
```

Create copy of the content structure above.

```
>>> CircleCopy = TXLWriter.AddContentStructure('MyCircleCopy')
>>> CircleCopy.AddPattern(
...     'Reference',
...     ReferencedStructureID=CircleStructure.ID,
...     OriginPoint=[40,60]
... )
<TXLWizard.Patterns.Reference.Reference object at 0x...>
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/Patterns/Reference')
```

ReferencedStructureID = None
> *str* – ID of the structure being referenced to

Type = None
> *str* – specifies the type of the pattern. Set to 'Reference'

4.2.8 TXLWizard.Patterns.Structure

Implements a class for *Structure* objects (*STRUCT*).

A *Structure* is a container for *Pattern* objects.

Classes

Structure(ID, **kwargs)	Implements a class for *Structure* objects.

class TXLWizard.Patterns.Structure.**Structure**(*ID*, ***kwargs*)
> Bases: *TXLWizard.Patterns.AbstractPattern.AbstractPattern*

> Implements a class for *Structure* objects.

> Corresponds to the TXL command *STRUCT*.

> A *Structure* is a container for *Pattern* objects.

>> **Parameters**

>> - **ID** (*str*) – Unique identification of the structure. Also used when referencing to this structure.

>> - **TXLOutput** (*bool, optional*) – If set to False, the TXL Output is suppressed.

>>   Defaults to True

>> - ****kwargs** – keyword arguments passed to the *TXLWizard.Patterns.AbstractPattern.AbstractPa* constructor. Can specify attributes of the current pattern.

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Create Content Structure

```
>>> CircleStructure = TXLWriter.AddContentStructure('MyCircleID',
...     TXLOutput = True
... )
>>> CircleStructure.AddPattern(
...     'Circle',
...     Center=[0, 0],
...     Radius=50,
...     Layer=1
... )
<TXLWizard.Patterns.Circle.Circle object at 0x...>
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/Patterns/Structure')
```

AddPattern(*PatternType*, ***kwargs*)
Adds a *Pattern* of type *PatternType* to the structure. Creates an instance of *TXLWizard.Patterns.{PatternType}.{PatternType}*. The *kwargs* are passed to the corresponding constructor and allow specifying pattern parameters as defined in the constructor of the corresponding pattern class and attributes as defined in *TXLWizard.Patterns.AbstractPattern.AbstractPattern*.

> **Parameters**
>
> - **PatternType** (*{'Array', 'Circle', 'Ellipse', 'Polygon', 'Polyline', 'Reference'}*) – Type of the pattern to be added.
> - ****kwargs** – keyword arguments are passed to the corresponding constructor and allow specifying pattern parameters as defined in the constructor of the corresponding pattern class and attributes as defined in *TXLWizard.Patterns.AbstractPattern.AbstractPattern*.
>
> **Returns** returns the created pattern object
>
> **Return type** TXLWizard.Patterns.{PatternType}.{PatternType}

CurrentAttributes = None
dict – attribute values of the next pattern to be added. Default values are copied from *self.DefaultAttributes*

ID = None
str – Unique identification of the structure. Also used when referencing to this structure.

Patterns = None
list of *TXLWizard.Patterns.AbstractPattern.AbstractPattern* – Patterns that are contained in this structure

TXLOutput = None
bool – If set to False, the TXL Output is suppressed.

Type = None
str – specifies the type of the pattern. Set to 'Structure'

4.3 Shape Library

[TXLWizard.ShapeLibrary.Label](#)	Renders arbitrary text in *TXLWriter*.
[TXLWizard.ShapeLibrary.LabelArray](#)	Renders an array of arbitrary text in *TXLWriter*.
[TXLWizard.ShapeLibrary.EndpointDetectionWindows](#)	Add five squares to *TXLWriter* that can be used as endpoint c
[TXLWizard.ShapeLibrary.AlignmentMarkers](#)	Add squares to *TXLWriter* that can be used as alignment mar

4.3.1 TXLWizard.ShapeLibrary.Label

Renders arbitrary text in *TXLWriter*.

Functions

[GetLabel](#)(TXLWriter, Text[, OriginPoint, ...])	Renders arbitrary text.

TXLWizard.ShapeLibrary.Label.**GetLabel**(*TXLWriter, Text, OriginPoint=[0, 0], FontSize=100, StrokeWidth=10, RotationAngle=0, FillCharacters=True, RoundCaps=False, Layer=1, **kwargs*)

Renders arbitrary text. Will have an automatically generated ID.

> **Parameters**
>
> - **TXLWriter** (*TXLWizard.TXLWriter.TXLWriter*) – Current Instance of *TXLWizard.TXLWriter.TXLWriter*
>
> - **Text** (*str*) – Text to be displayed
>
> - **OriginPoint** (*list of float, optional*) – x- and y-coordinates of the origin point of the label. Defaults to [0,0]
>
> - **FontSize** (*float, optional*) – Font size. Character height = font size. Defaults to 100
>
> - **StrokeWidth** (*float*) – line thickness of the letters. Defaults to 10
>
> - **RotationAngle** (*float*) – Angle by which the text is rotated. Defaults to 0
>
> - **FillCharacters** (*bool, optional*) – If set to True, closed boundaries will be filled. Can be useful if there should be no free-standing parts. Defaults to True
>
> - **RoundCaps** (*bool, optional*) – If set to True, the paths will habe rounded ends. Should be set to False for better e-Beam Performance Defaults to False.
>
> - **Layer** (*int, optional*) – Layer the text should be rendered in. Defaults to 1
>
> - ****kwargs** – keyword arguments
>
> **Returns** *Structure* object containing the patterns representing the text
>
> **Return type** [*TXLWizard.Patterns.Structure.Structure*](#)

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
>>> import TXLWizard.ShapeLibrary.Label
```

Initialize TXLWriter >>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()

Add a Label

```
>>> SampleLabelObject = TXLWizard.ShapeLibrary.Label.GetLabel(
...     TXLWriter,
...     Text='This is my text',
...     OriginPoint=[
...         -200, 300
...     ],
...     FontSize=150,
...     StrokeWidth=20,
...     RoundCaps=False, # Set to False to improve e-Beam performance
...     Layer=1
... )
```

Generate files

```
>>> TXLWriter.GenerateFiles('Tests/Results/ShapeLibrary/Label')
```

4.3.2 TXLWizard.ShapeLibrary.LabelArray

Renders an array of arbitrary text in *TXLWriter*.

Functions

GetLabelArray(TXLWriter, Text, OriginPoint, ...)	Renders an array of arbitrary text.

TXLWizard.ShapeLibrary.LabelArray.**GetLabelArray**(*TXLWriter*, *Text*, *OriginPoint*, *PositionDelta1*, *PositionDelta2*, *Repetitions1*, *Repetitions2*, *FontSize=100*, *StrokeWidth=10*, *RotationAngle=0*, *FillCharacters=True*, *RoundCaps=False*, *Layer=1*, ***kwargs*)

Renders an array of arbitrary text. Will have an automatically generated ID. The markers *{i}* and *{j}* in the text are substituted with an auto-incremented index starting at 1.

Parameters

- **TXLWriter** (*TXLWizard.TXLWriter.TXLWriter*) – Current Instance of *TXLWizard.TXLWriter.TXLWriter*

- **Text** (*str*) – Text to be displayed

- **OriginPoint** (*list of float*) – x- and y-coordinates of the origin point of the label array.

- **PositionDelta1** (*list of float*) – x- and y- coordinates of the first replication direction.

- **PositionDelta2** (*list of float*) – x- and y- coordinates of the second replication direction.

- **Repetitions1** (*int*) – Number of replications in the first replication direction

- **Repetitions2** (`int`) – Number of replications in the second replication direction

- **FontSize** (`float, optional`) – Font size. Character height = font size.

  Defaults to 100

- **StrokeWidth** (`float, optional`) – line thickness of the letters.

  Defaults to 10

- **RotationAngle** (`float, optional`) – Angle by which the text is rotated.

  Defaults to 0

- **FillCharacters** (`bool, optional`) – If set to True, closed boundaries will be filled. Can be useful if there should be no free-standing parts.

  Defaults to True

- **RoundCaps** (`bool, optional`) – If set to True, the paths will habe rounded ends. Should be set to False for better e-Beam Performance.

  Defaults to False.

- **Layer** (`int, optional`) – Layer the text should be rendered in.

  Defaults to 1

- ****kwargs** – keyword arguments

Returns *Structure* object containing the patterns representing the text

Return type *TXLWizard.Patterns.Structure.Structure*

Examples

Import required modules

```
>>> import TXLWizard.TXLWriter
>>> import TXLWizard.ShapeLibrary.LabelArray
```

Initialize TXLWriter

```
>>> TXLWriter = TXLWizard.TXLWriter.TXLWriter()
```

Add a label array

```
>>> SampleLabelObject = TXLWizard.ShapeLibrary.LabelArray.GetLabelArray(
...     TXLWriter,
...     Text='ObjectA_A{i}B{j}',
...     OriginPoint=[
...         -200, 300
...     ],
...     PositionDelta1=[
...         500, 0
...     ],
...     PositionDelta2=[
...         0, 100
...     ],
...     Repetitions1=3,
...     Repetitions2=5,
...     FontSize=20,
...     StrokeWidth=4,
```

```
...         RoundCaps=True, # Set to False to improve e-Beam performance
...         Layer=1
... )
```

Generate Files

```
>>> TXLWriter.GenerateFiles('Tests/Results/ShapeLibrary/LabelArray')
```

4.3.3 TXLWizard.ShapeLibrary.EndpointDetectionWindows

Add five squares to *TXLWriter* that can be used as endpoint detection windows.

Functions

GetEndpointDetectionWindows(TXLWriter[, ...])	Add five squares that can be used as endpoint detection windows.

TXLWizard.ShapeLibrary.EndpointDetectionWindows.**GetEndpointDetectionWindows**(*TXLWriter*, *SizeLarge=1000*, *SizeSmall=750*, *Offset=1500*, *Layer=1*)

Add five squares that can be used as endpoint detection windows. The first square of size *SizeLarge* will be placed in the center. The second to fifth square of size *SizeSmall* will be placed at x / y = +-*Offset* / +-*Offset*

> **Parameters**
>
> - **TXLWriter** (*TXLWizard.TXLWriter.TXLWriter*) – Current Instance of *TXLWizard.TXLWriter.TXLWriter*
>
> - **SizeLarge** (*float, optional*) – Size of the center square.
>
>   Defaults to 1000
>
> - **SizeSmall** (*float, optional*) – Size of the four peripheral square.
>
>   Defaults to 750
>
> - **Offset** (*float, optional*) – Offset of the peripheral squares to the center.
>
>   Defaults to 1500
>
> - **Layer** (*int, optional*) – Layer the pattern should be rendered in.
>
>   Defaults to 1
>
> **Returns** *Structure* object containing the patterns representing the endpoint detection windows
>
> **Return type** *TXLWizard.Patterns.Structure.Structure*

4.3.4 TXLWizard.ShapeLibrary.AlignmentMarkers

Add squares to *TXLWriter* that can be used as alignment markers.

Functions

[*GetAlignmentMarkers*](TXLWriter[, Size, ...])	Add squares that can be used as alignment markers

TXLWizard.ShapeLibrary.AlignmentMarkers.**GetAlignmentMarkers** (*TXLWriter*, *Size=10*, *OffsetSmall=750*, *OffsetLarge=3000*, *Layer=1*)

> Add squares that can be used as alignment markers

> > **Parameters**

> > - **TXLWriter** (*TXLWizard.TXLWriter.TXLWriter*) – Current Instance of *TXLWizard.TXLWriter.TXLWriter*

> > - **Size** (*float, optional*) – Size of the markers.

> > > Defaults to 10

> > - **OffsetSmall** (*float, optional*) – first offset from center.

> > > Defaults to 750

> > - **OffsetLarge** (*float, optional*) – second offset from center.

> > > Defaults to 3000

> > - **Layer** (*int, optional*) – Layer the pattern should be rendered in.

> > > Defaults to 1

> > **Returns** *Structure* object containing the patterns representing the alignment markers

> > **Return type** *TXLWizard.Patterns.Structure.Structure*

4.4 TXLConverter

[*TXLWizard.TXLConverter*](#)	Class for parsing TXL files and converting them to html / svg using *TXLWriter*

4.4.1 TXLWizard.TXLConverter

Class for parsing TXL files and converting them to html / svg using *TXLWriter*

Classes

[*TXLConverter*](Filename, **kwargs)	Class for parsing TXL files and converting them to
[*TXLConverterCLI*]([JSONConfigurationFile, ...])	Provides a command line interface for the *TXLConverter* class.

class TXLWizard.TXLConverter.**TXLConverter** (*Filename*, ***kwargs*)

> Bases: object

> Class for parsing TXL files and converting them to html / svg using *TXLWizard.TXLWriter*

> > **Parameters**

- **Filename** (`str`) – Path / Filename of the .txl file

- **LayersToProcess** (`list of int, optional`) – if given, only layers in this list are processed / shown

- ****kwargs** – keyword-arguments passed to the *TXLWizard.TXLWriter.TXLWriter* constructor.

Examples

Import required modules

```
>>> import TXLWizard.TXLConverter
```

Instatiate a *TXLConverter* instance, parse the TXL file and generate the HTML / SVG files. Saves the converted files to *Tests/Results/TXLConverter/*

```
>>> TXLConverterInstance = TXLWizard.TXLConverter.TXLConverter(
...     'Tests/SampleFiles/Example_Advanced_Original.txl',
...     GridWidth=500,
...     GridHeight=800,
...     LayersToProcess=[1,2,3,5]
... )
>>> TXLConverterInstance.ParseTXLFile()
>>> TXLConverterInstance.GenerateFiles('Tests/Results/TXLConverter')
```

GenerateFiles(*TargetFolder=None*)
> Generate the HTML / SVG files

>> **Parameters** **TargetFolder** (`str, optional`) – If given, the converted files are stored in the folder specified.

>> If not given, the converted files are stored in the same folder as the original file.

>> Defaults to None.

ParseTXLFile()
> Parses the TXL file by processing line by line. The number of references to structures is counted.

class TXLWizard.TXLConverter.**TXLConverterCLI**(*JSONConfigurationFile='TXLConverterConfiguration.json'*, *UpdateConfigurationFile=True*, *OverrideConfiguration={}*)

> Bases: `object`

> Provides a command line interface for the *TXLConverter* class.

> The configuration is read and stored in the JSON format in the file specified in *JSONConfigurationFile*.

>> **Parameters**

>> - **JSONConfigurationFile** (`str, optional`) – Path / Filename of the file where the configuration is read and stored in the JSON format. Defaults to 'TXLConverterConfiguration.json'

>> - **UpdateConfigurationFile** (`bool, optional`) – Flag whether to update the configuration file. Defaults to True.

>> - **OverrideConfiguration** (`dict, optional`) – Dictionary with configuration options overriding the default / stored configuration. Defaults to {}

Examples

Import required modules

```
>>> import TXLWizard.TXLConverter
```

Start the command line interface

```
>>> TXLWizard.TXLConverter.TXLConverterCLI()
```

t