TXLWizard Documentation

Release 1.7.0

Esteban Marin

Jun 07, 2016

TABLE OF CONTENTS

CHAPTER

ONE

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:

```
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:

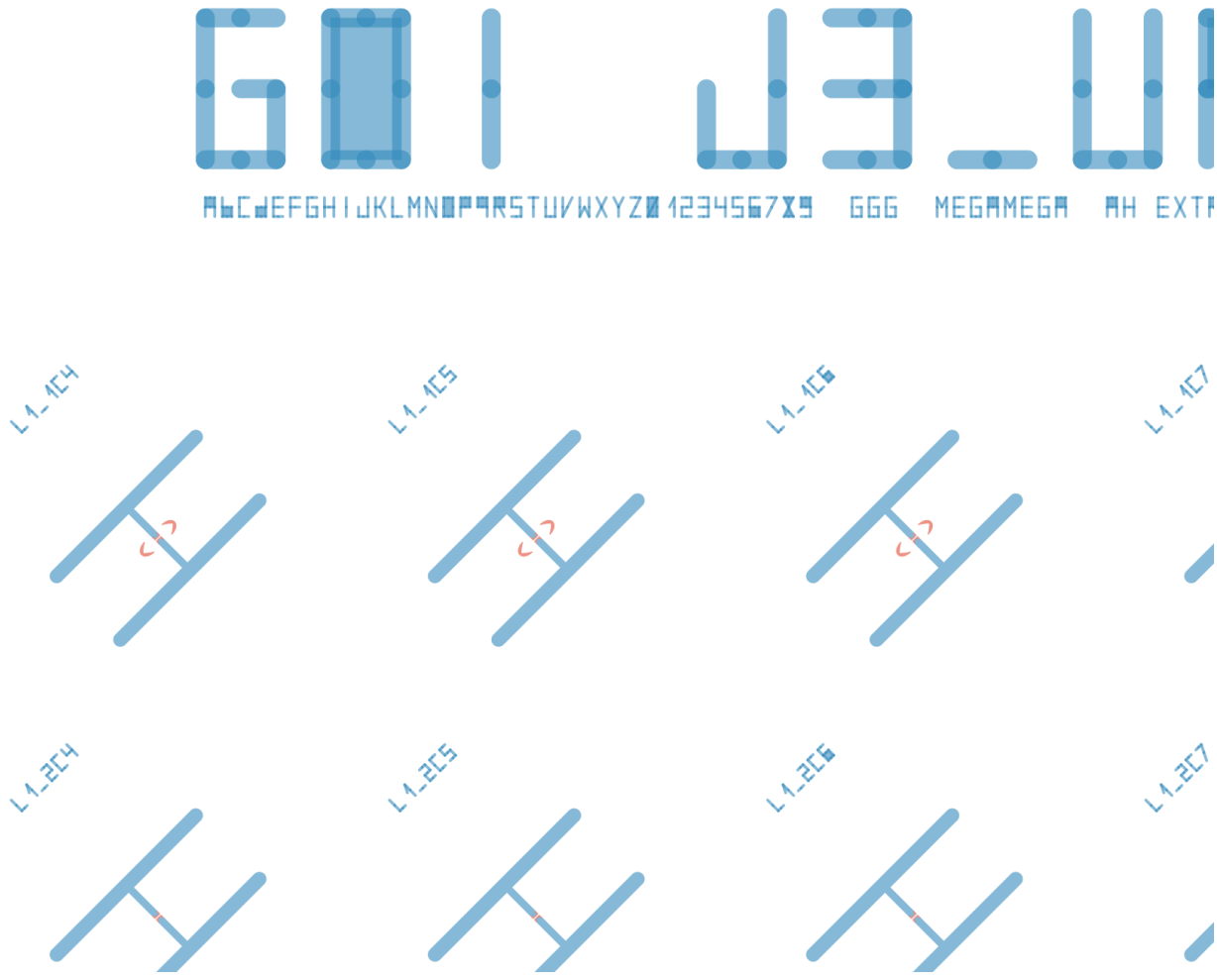

Fig. 1.1: Example SVG output for a mask

1.6 How to start?

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

CHAPTER

TWO

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

```
##########################################
# 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
```

```
    Layer=1
)

## User Structure: Circle ##

# Create Content Structure for Circle with ID `MyCircle`
CircleStructure = TXLWriter.AddContentStructure('MyCircle')

# Add a `Pattern` of type `Circle`
CircleStructure.AddPattern(
    'Circle',
    Center=[0, 0],
    Radius=150,
    Layer=2
)

#########################
# Generate Output Files #
#########################

# Note: The suffix (.txl, .html, .svg) will be appended automatically
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

```
####################
# Import Libraries #
####################

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

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

# Import math module for calculations
import math

################################
```

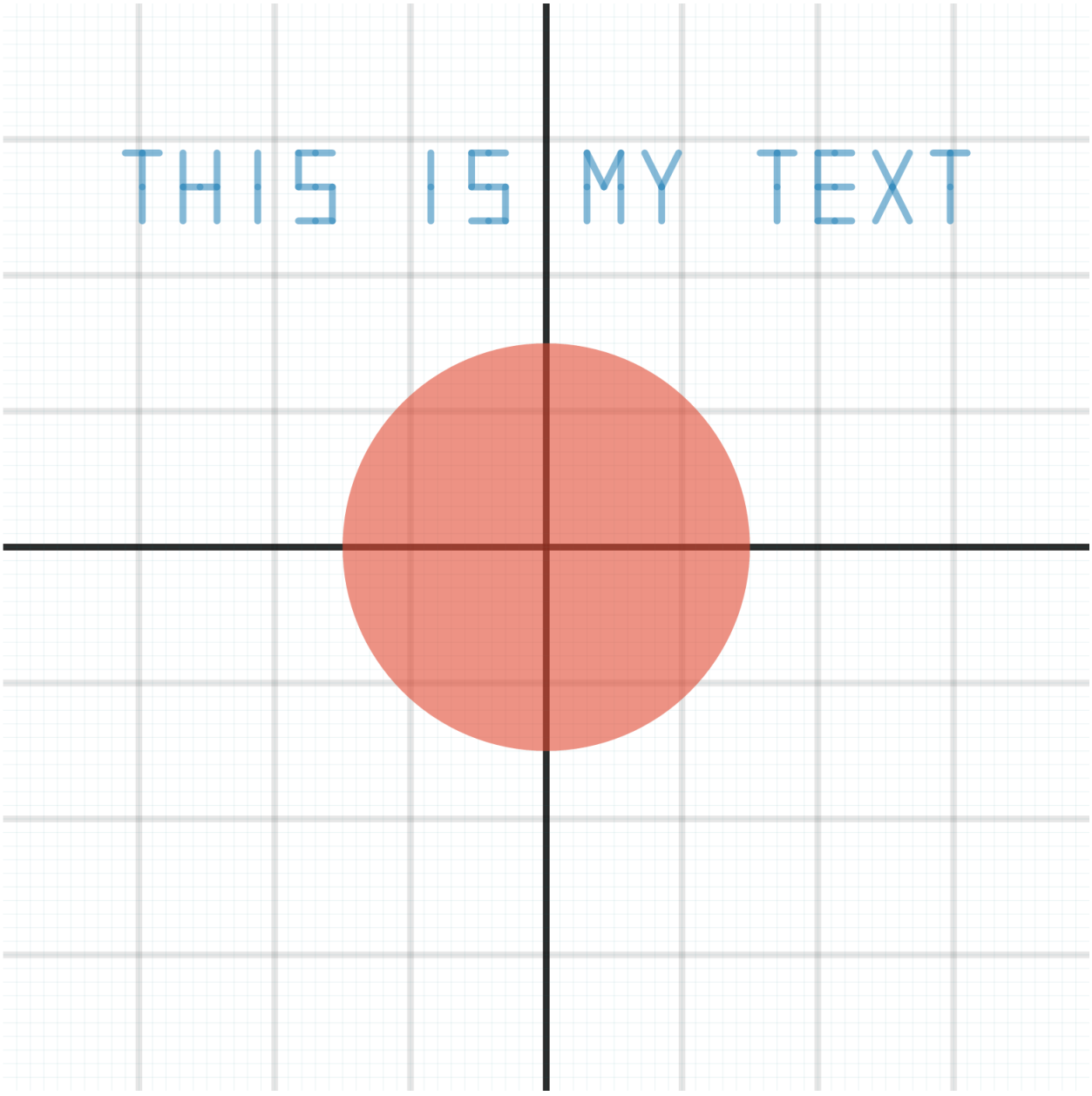

Fig. 2.1: Generated SVG Image for *Content/Example_Introduction.py*

```
# Sample / Structure Parameters #
################################

# Define all sample parameters
SampleParameters = {
    'Width': 8e3,
    'Height': 8e3,
    'Label': 'Simple Demo',
}

# Define all structure parameters
StructureParameters = {
    'Circle': {
        'Radius': 50,
        'Layer': 3
    },
    'CircleArray': {
        'Columns': 6,
        'Rows': 5,
        'ArrayXOffset': 500,
        'ArrayYOffset': -500,
        'ArrayOrigin': [0.75e3, 3e3],
        'Label': 'R{i}C{j}',
    }
}

########################
# Initialize TXLWriter #
########################
TXLWriter = TXLWizard.TXLWriter.TXLWriter(
    GridWidth=SampleParameters['Width'],
    GridHeight=SampleParameters['Height']
)

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

## Sample Label ##

# Give the sample a nice label
SampleLabelObject = TXLWizard.ShapeLibrary.Label.GetLabel(
    TXLWriter,
    Text=SampleParameters['Label'],
    OriginPoint=[
        0.5e3, 1. * SampleParameters['Height'] / 2. - 500
    ],
    FontSize=150,
    StrokeWidth=20,
    RoundCaps=True,  # Set to False to improve e-Beam performance
    Layer=1
)

## Endpoint Detection ##

# Use Pre-Defined Endpoint Detection Windows
TXLWizard.ShapeLibrary.EndpointDetectionWindows.GetEndpointDetectionWindows(
    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 #
```

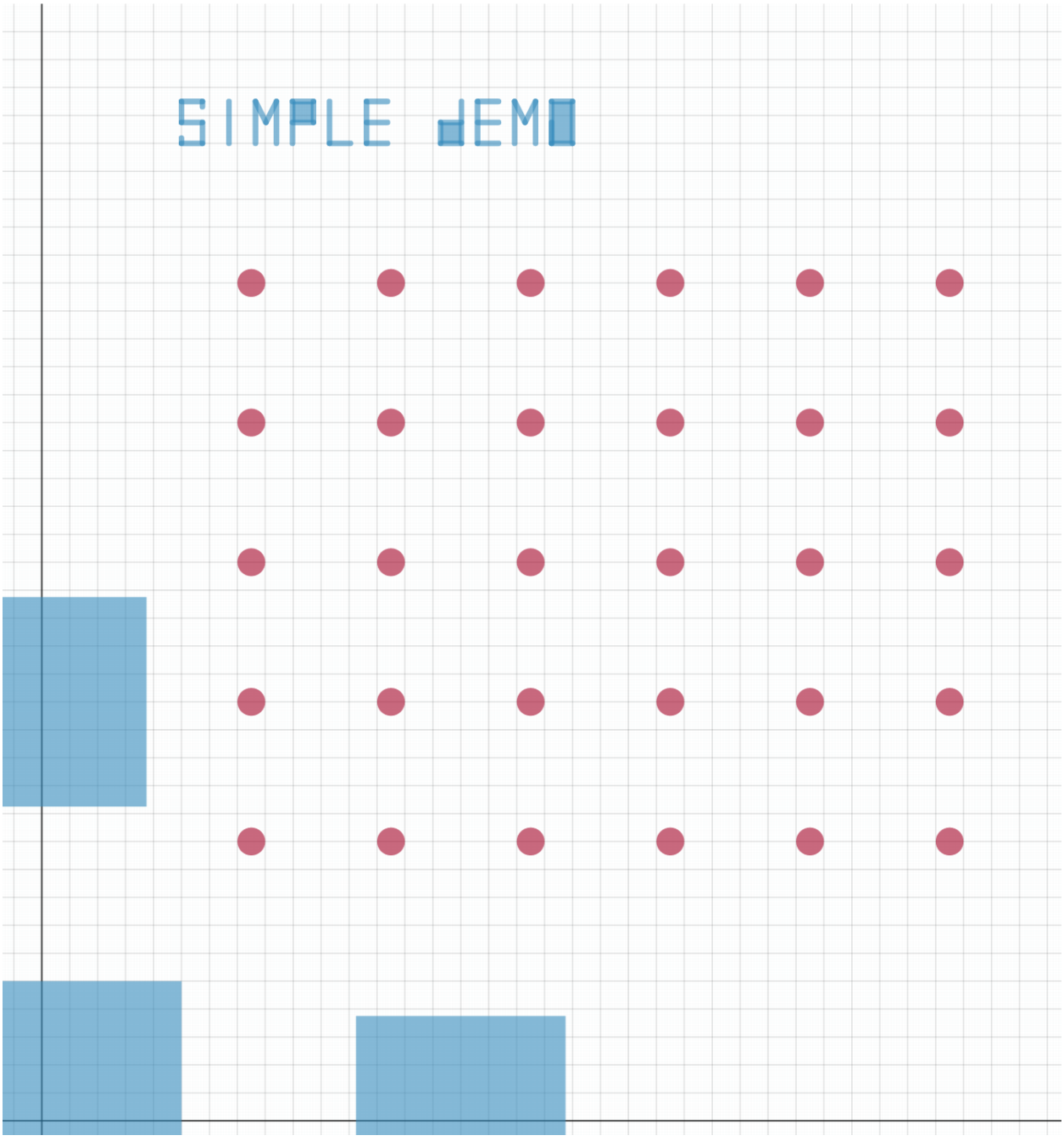

Fig. 2.2: Generated SVG Image for *Content/Example_Simple.py*

```
####################

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

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

################################
# Sample / Structure Parameters #
################################

# Define all sample parameters
SampleParameters = {
    'Width': 8e3,
    'Height': 8e3,
    'Label': 'Simple Demo',
}

# Define all structure parameters
StructureParameters = {
    'CircleArray': {
        'Columns': 6,
        'Rows': 5,
        'ArrayXOffset': 500,
        'ArrayYOffset': -500,
        'ArrayOrigin': [0.75e3, 3e3],
        'Label': 'R{j}C{i}',  # {i} and {j} will be replaced
        # by str.format() with the corresponding auto-incremented index
        'LabelXOffset': 0,
        'LabelYOffset': 100,
    }
}

########################
# Initialize TXLWriter #
########################
TXLWriter = TXLWizard.TXLWriter.TXLWriter(
    GridWidth=SampleParameters['Width'],
    GridHeight=SampleParameters['Height']
)

# Import existing TXL file
TXLWriter.ImportTXLFile('Masks/Example_Simple.txl')

# label each array element
TXLWizard.ShapeLibrary.LabelArray.GetLabelArray(
    TXLWriter,
    StructureParameters['CircleArray']['Label'],
    OriginPoint=[
        StructureParameters['CircleArray']['ArrayOrigin'][0]
        + StructureParameters['CircleArray']['LabelXOffset'],
        StructureParameters['CircleArray']['ArrayOrigin'][1]
        + StructureParameters['CircleArray']['LabelYOffset']
    ],
    PositionDelta1=[
        StructureParameters['CircleArray']['ArrayXOffset'], 0
    ],
```

```
    PositionDelta2=[
        0, StructureParameters['CircleArray']['ArrayYOffset']
    ],
    Repetitions1=StructureParameters['CircleArray']['Columns'],
    Repetitions2=StructureParameters['CircleArray']['Rows']
)

#########################
# Generate Output Files #
#########################

# Note: The suffix (.txl, .html, .svg) will be appended automatically
TXLWriter.GenerateFiles('Masks/Example_ImportTXLFile')
```

2.3.3 Generated SVG Image

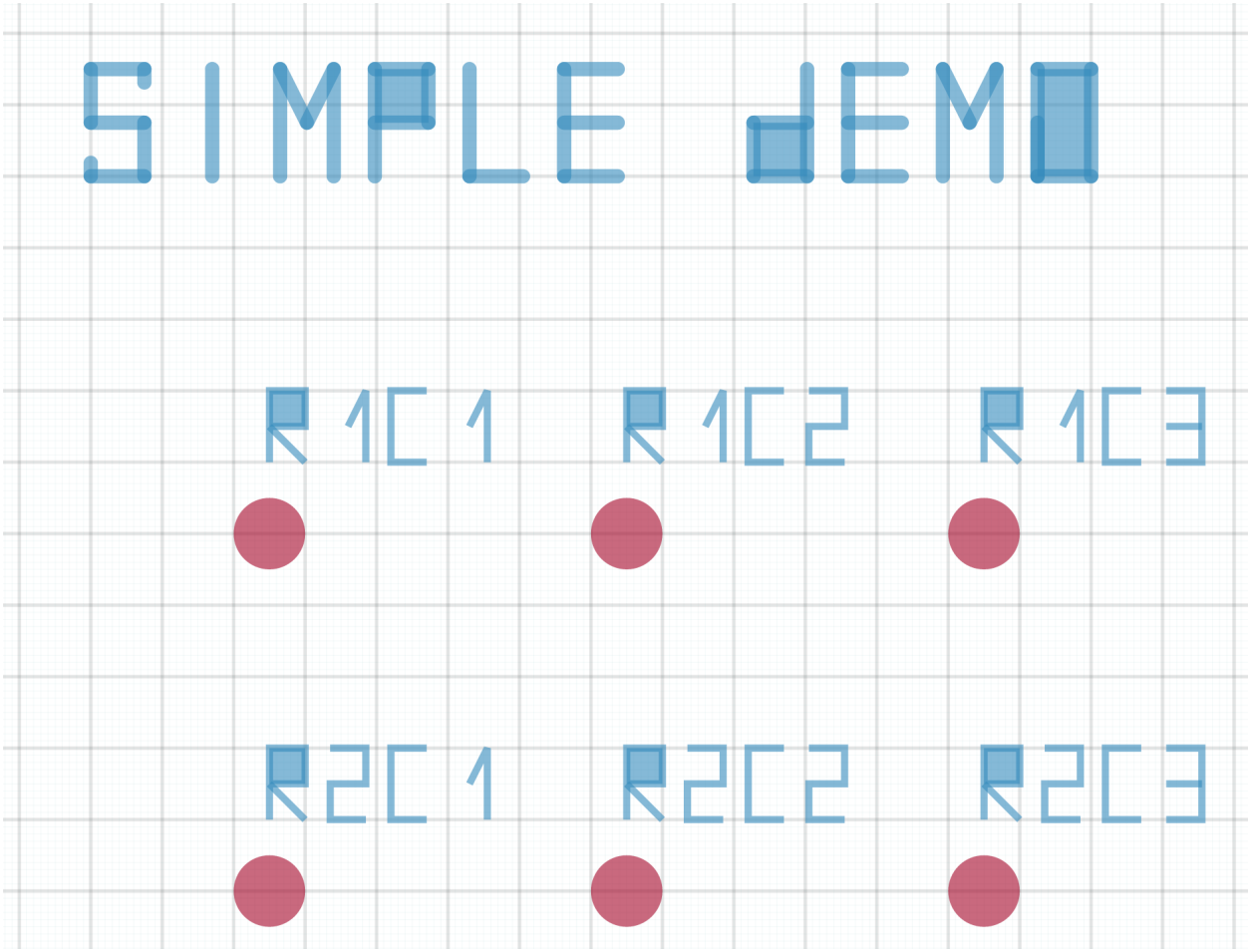

Fig. 2.3: Generated SVG Image for *Content/Example_ImportTXLFile.py*

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

```
####################
# Import Libraries #
####################

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

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

# Import math module for calculations
import math


################################
# Sample / Structure Parameters #
################################

# Define all sample parameters
SampleParameters = {
    'Width': 8e3,
    'Height': 8e3,
    'Label': 'GOI Demo CornerCube',
}

# Define all structure parameters
StructureParameters = {
    'CornerCube': {
        'BridgeLength':40,
        'ParabolaFocus': 45,
        'XCutoff': 45,
        'AirGapX': 15,
        'AirGapY': 5,
        'LabelXOffset': 0,
        'LabelYOffset': 150,
        'Label': 'R{i}C{j}', # {i} and {j} will be replaced
                             # by str.format() with the corresponding row / column index
        'Layer': 2
    },
    'Circle': {
```

```
        'Radius': 25,
        'Layer': 3
    },
    'CornerCubeArray': {
        'Columns': 6,
        'Rows': 3,
        'ArrayXOffset': 500,
        'ArrayYOffset': -500,
        'ArrayOrigin': [0.75e3, 3e3]
    }
}


########################
# Initialize TXLWriter #
########################
TXLWriter = TXLWizard.TXLWriter.TXLWriter(
    GridWidth=SampleParameters['Width'],
    GridHeight=SampleParameters['Height'],
    Precision=6 #increase the precision / resolution to 0.000001 (10^-6)
)

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

## Sample Label ##

# Give the sample a nice label...
SampleLabelObject = TXLWizard.ShapeLibrary.Label.GetLabel(
    TXLWriter,
    Text=SampleParameters['Label'],
    OriginPoint=[
        0.5e3, 1. * SampleParameters['Height'] / 2. - 500
    ],
    FontSize=150,
    StrokeWidth=20,
    RoundCaps=False,# Set to False to improve e-Beam performance
    Layer=1
)
# ...and some other information
Alphabet = TXLWizard.ShapeLibrary.Label.GetLabel(
    TXLWriter,
    Text='abcdefghijklmnopqrstuvwxyz0123456789 megamega ggg ah extraaaa rischaaaar',
    OriginPoint=[
        0.5e3, 1. * SampleParameters['Height'] / 2. - 600
    ],
    FontSize=50,
    StrokeWidth=3,
    RoundCaps=False, # Set to False to improve e-Beam performance
    Layer=1
)

## Endpoint Detection ##

# Use Pre-Defined Endpoint Detection Windows
TXLWizard.ShapeLibrary.EndpointDetectionWindows.GetEndpointDetectionWindows(
    TXLWriter, Layer=1
```

```
)

## Alignment Markers ##

# Use Pre-Defined Alignment Markers
TXLWizard.ShapeLibrary.AlignmentMarkers.GetAlignmentMarkers(
    TXLWriter, Layer=1
)


## User Structure: Corner Cube ##

# Create Definition Structure for Corner Cube that will be reused
CornerCubeDefinition = TXLWizard.ShapeLibrary.CornerCube.GetCornerCube(
    TXLWriter,
    ParabolaFocus=StructureParameters['CornerCube']['ParabolaFocus'],
    XCutoff=StructureParameters['CornerCube']['XCutoff'],
    AirGapX=StructureParameters['CornerCube']['AirGapX'],
    AirGapY=StructureParameters['CornerCube']['AirGapY'],
    Layer=StructureParameters['CornerCube']['Layer']
)

# Create Definition Structure for combination of cornercube and additional circle
FullCornerCubeNoRotation = TXLWriter.AddDefinitionStructure('FullCornerCubeNoRotation')
FullCornerCubeNoRotation.AddPattern(
    'Reference',
    ReferencedStructureID=CornerCubeDefinition.ID,
    OriginPoint=[1. * StructureParameters['CornerCube']['BridgeLength'] / 2., 0]
)
FullCornerCubeNoRotation.AddPattern(
    'Circle',
    Center=[0, 0],
    Radius=StructureParameters['Circle']['Radius'],
    Layer=StructureParameters['Circle']['Layer']
)

# Create definition structure with rotation of entire referenced structure
FullCornerCube = TXLWriter.AddDefinitionStructure('FullCornerCube',
                                                  RotationAngle=45)
FullCornerCube.AddPattern(
    'Reference',
    ReferencedStructureID=FullCornerCubeNoRotation.ID,
    OriginPoint=[0, 0]
)

# Create array of the definition structure above
CornerCubeArrayFine = TXLWriter.AddContentStructure('CornerCubeArrayFine')
CornerCubeArrayFine.AddPattern(
    'Array',
    ReferencedStructureID=FullCornerCube.ID,
    OriginPoint=StructureParameters['CornerCubeArray']['ArrayOrigin'],
    PositionDelta1=[
        StructureParameters['CornerCubeArray']['ArrayXOffset'], 0
    ],
    PositionDelta2=[
        0, StructureParameters['CornerCubeArray']['ArrayYOffset']
    ],
    Repetitions1=StructureParameters['CornerCubeArray']['Columns'],
```

```
    Repetitions2=StructureParameters['CornerCubeArray']['Rows']
)

# Add a label array to label each element of the array pattern above
TXLWizard.ShapeLibrary.LabelArray.GetLabelArray(
    TXLWriter,
    StructureParameters['CornerCube']['Label'],
    OriginPoint=[
        StructureParameters['CornerCubeArray']['ArrayOrigin'][0]
        + StructureParameters['CornerCube']['LabelXOffset'],
        StructureParameters['CornerCubeArray']['ArrayOrigin'][1]
        + StructureParameters['CornerCube']['LabelYOffset']
    ],
    PositionDelta1=[
        StructureParameters['CornerCubeArray']['ArrayXOffset'], 0
    ],
    PositionDelta2=[
        0, StructureParameters['CornerCubeArray']['ArrayYOffset']
    ],
    Repetitions1=StructureParameters['CornerCubeArray']['Columns'],
    Repetitions2=StructureParameters['CornerCubeArray']['Rows'],
    FontSize=40,
    StrokeWidth=3,
    RoundCaps=False,# Set to False to improve e-Beam performance
    Layer=1,
    RotationAngle=45
)



#########################
# Generate Output Files #
#########################

# Note: The suffix (.txl, .html, .svg) will be appended automatically
TXLWriter.GenerateFiles('Masks/Example_Advanced')

```

2.4.3 Generated SVG Image

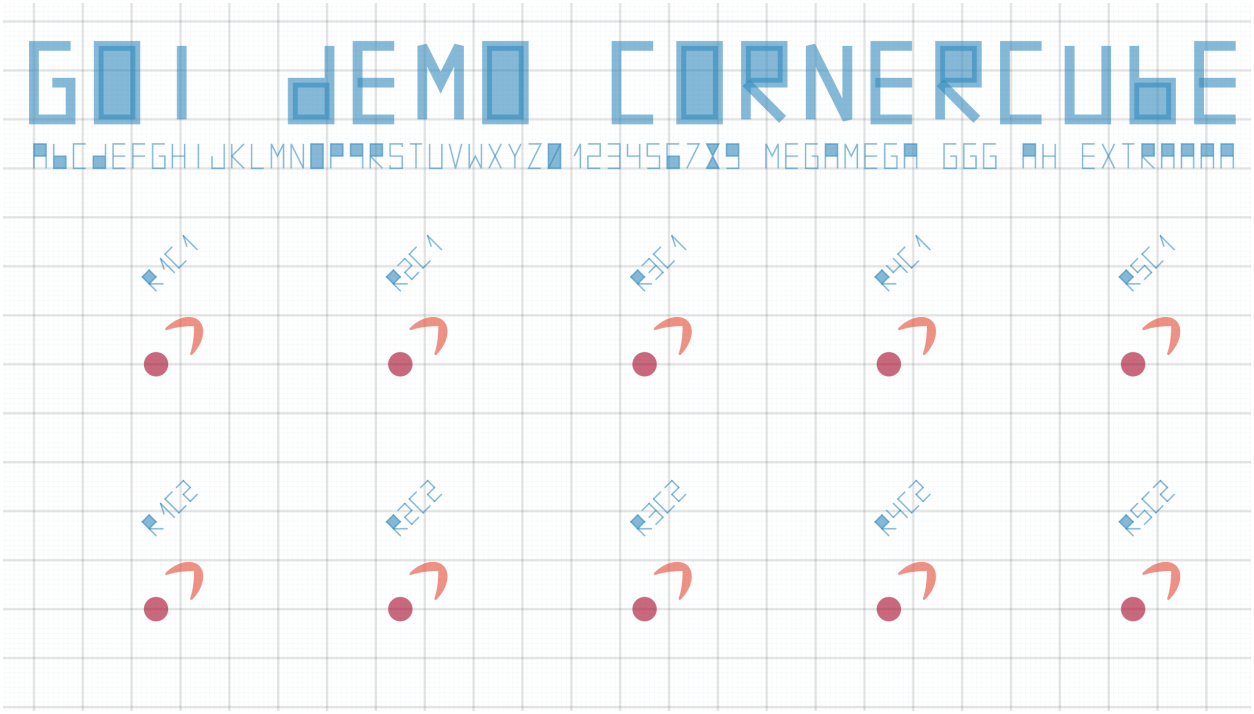

Fig. 2.4: Generated SVG Image for *Content/Example_Advanced.py*

CHAPTER

THREE

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

```
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
```

CHAPTER

FOUR

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

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)
```

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]
>>> )
```

Generate the Output files with name *mask.(txl|html|svg)* to the folder *myPath*

```
>>> TXLWriter.GenerateFiles('myPath/mask')
```

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*)

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

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

Initialize TXLWriter

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

import TXL file *myPath/mask_orig.txl*

```
>>> TXLWriter.ImportTXLFile('myPath/mask_orig.txl', LayersToProcess=[2,4])
```

Add a pattern to an existing structure

```
>>> MyStructure = TXLWriter._ContentStructures['MySuperStructure']
>>> MyStructure.AddPattern(
>>>     'Circle',
>>>     Center=[0, 0],
>>>     Radius=50,
>>>     Layer=1
>>> )
```

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 *mask_final.(txl\html\svg)* to the folder *myPath*

```
>>> TXLWriter.GenerateFiles('myPath/mask_final')
```

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

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
>>> )
```

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
>>> )
```

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

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
>>> )
```

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

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
>>> )
```

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

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
>>> )
```

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
>>> )
```

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

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
>>> )
```

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
>>> )
```

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

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
>>> )
```

Create copy of the content structure above.

```
>>> CircleCopy = TXLWriter.AddContentStructure('MyCircleCopy')
>>> CircleCopy.AddPattern(
>>>     'Reference',
>>>     ReferencedStructureID=CircleStructure.ID,
>>>     OriginPoint=[40,60]
>>> )
```

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

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
>>> )
```

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.EndpointDetectionWindows	Add five squares to *TXLWriter* that can be used as endpoint d
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

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

4.3.2 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*

4.3.3 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

Instatiate a *TXLConverter* instance, parse the TXL file and generate the HTML / SVG files

```
>>> TXLConverterInstance = TXLWizard.TXLConverter.TXLConverter(
>>>     FullFilePath,
>>>     GridWidth=500,
>>>     GridHeight=800,
>>>     LayersToProcess=[1,5]
>>> )
>>> TXLConverterInstance.ParseTXLFile()
>>> TXLConverterInstance.GenerateFiles()
```

GenerateFiles()

Generate the HTML / SVG files

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

Start the command line interface

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

t