RoboScribe

1.0

Generated by Doxygen 1.10.0

1 Namespace Documentation 1

1.1 main Namespace Reference . . . 1

1.1.1 Detailed Description . . . 1

1.1.2 Function Documentation . . . 2

1.1.2.1 generuj_gcode_dla_punktow() . . . 2

1.1.2.2 probkowanie() . . . 2

1.1.2.3 przetwarzaj_circle() . . . 2

1.1.2.4 przetwarzaj_ellipse() . . . 3

1.1.2.5 przetwarzaj_line() . . . 3

1.1.2.6 przetwarzaj_polygon() . . . 3

1.1.2.7 przetwarzaj_polyline() . . . 3

1.1.2.8 przetwarzaj_rect() . . . 4

1.1.3 Variable Documentation . . . 4

1.1.3.1 gcode_wyjsciowy . . . 4

1.1.3.2 punkty . . . 4

1.1.3.3 Rozdzielczosc . . . 4

1.1.3.4 skala . . . 4

1.1.3.5 svg . . . 4

2 File Documentation 5

2.1 main.py File Reference . . . 5

2.2 main.py . . . 6

Index 9

Chapter 1

Namespace Documentation

1.1 main Namespace Reference

Functions

- probkowanie (path, rozdzielczosc)
- przetwarzaj_rect (element)
- przetwarzaj_circle (element)
- przetwarzaj_ellipse (element)
- przetwarzaj_line (element)
- przetwarzaj_polyline (element)
- przetwarzaj_polygon (element)
- generuj_gcode_dla_punktow (punkty)

Variables

- float skala = 1.0

  Skala konwersji współrzędnych SVG na jednostki G-code.
- int Rozdzielczosc = 1

  Rozdzielczość próbkowania - maksymalna odległość między punktami.
- svg = SVG.parse("output_path_arial.svg")

  Wczytana zawartość pliku SVG.
- list gcode_wyjsciowy = [ ]

  Lista poleceń G-code do zapisania do pliku.
- list punkty = [ ]

1.1.1 Detailed Description

```
@file main.py
@brief Główny moduł konwersji SVG do G-code
@details Konwertuje elementy SVG (prostokąty, okręgi, elipsy, linie, ścieżki)
         na sekwencje poleceń G-code dla urządzeń CNC/maszyn rysujących.
@author RoboScribe
@version 1.0
```

1.1.2 Function Documentation

1.1.2.1 generuj_gcode_dla_punktow()

```
main.generuj_gcode_dla_punktow (
              punkty )
```

Definition at line 143 of file main.py.

```
00143 def generuj_gcode_dla_punktow(punkty):
00144
00148     if not punkty:
00149         return
00150
00151     # Przejście do pierwszego punktu bez rysowania
00152     x0, y0 = punkty[0]
00153     x_val = round(x0 * skala, 2)
00154     y_val = round(y0 * skala, 2)
00155     gcode_wyjsciowy.append(f"G0 X{x_val} Y{y_val}")
00156
00157     gcode_wyjsciowy.append('G0 Z0')  # Opuszczenie pisaka
00158
00159     # Rysowanie pozostałych punktów
00160     for x, y in punkty[1:]:
00161         x_val = round(x * skala, 2)
00162         y_val = round(y * skala, 2)
00163         gcode_wyjsciowy.append(f"G1 X{x_val} Y{y_val}")
00164
```

1.1.2.2 probkowanie()

```
main.probkowanie (
              path,
              rozdzielczosc )
```

Definition at line 30 of file main.py.

```
00030 def probkowanie(path, rozdzielczosc):
00031
00035     punkty = []
00036     dlugosc_sciezki = path.length()
00037     #TODO: rozważyć jak ma działać rozdzielczość
00038     #n = max(1, int(dlugosc_sciezki / rozdzielczosc))
00039     n = int(dlugosc_sciezki / rozdzielczosc)
00040
00041     for i in range(n+1):
00042         p = path.point(i / n)
00043         punkty.append((p.real, p.imag))
00044     return punkty
00045
00046
```

1.1.2.3 przetwarzaj_circle()

```
main.przetwarzaj_circle (
              element )
```

Definition at line 69 of file main.py.

```
00069 def przetwarzaj_circle(element):
00070
00074     cx = element.cx or 0
00075     cy = element.cy or 0
00076     r = element.r or 0
00077
00078     # Liczba segmentów na podstawie Rozdzielczości
00079     segments = max(int(2 * math.pi * r / Rozdzielczosc), 8)
00080
00081     punkty = []
00082     for i in range(segments + 1):
00083         angle = (i / segments) * 2 * math.pi
00084         x = cx + r * math.cos(angle)
00085         y = cy + r * math.sin(angle)
00086         punkty.append((x, y))
00087
00088     return punkty
00089
```

1.1.2.4 przetwarzaj_ellipse()

```
main.przetwarzaj_ellipse (
                element )
```

Definition at line 90 of file main.py.

```
00090 def przetwarzaj_ellipse(element):
00091
00095     cx = element.cx or 0
00096     cy = element.cy or 0
00097     rx = element.rx or 0
00098     ry = element.ry or 0
00099
00100     # Liczba segmentów na podstawie średniej Rozdzielczości
00101     segments = max(int(2 * math.pi * max(rx, ry) / Rozdzielczosc), 8)
00102
00103     punkty = []
00104     for i in range(segments + 1):
00105         angle = (i / segments) * 2 * math.pi
00106         x = cx + rx * math.cos(angle)
00107         y = cy + ry * math.sin(angle)
00108         punkty.append((x, y))
00109
00110     return punkty
00111
```

1.1.2.5 przetwarzaj_line()

```
main.przetwarzaj_line (
                element )
```

Definition at line 112 of file main.py.

```
00112 def przetwarzaj_line(element):
00113
00116     x1 = element.x1 or 0
00117     y1 = element.y1 or 0
00118     x2 = element.x2 or 0
00119     y2 = element.y2 or 0
00120
00121     return [(x1, y1), (x2, y2)]
00122
```

1.1.2.6 przetwarzaj_polygon()

```
main.przetwarzaj_polygon (
                element )
```

Definition at line 131 of file main.py.

```
00131 def przetwarzaj_polygon(element):
00132
00136     if hasattr(element, 'points') and element.points:
00137         punkty = list(element.points)
00138         if punkty and punkty[0] != punkty[-1]:
00139             punkty.append(punkty[0])  # zamknięcie
00140         return punkty
00141     return []
00142
```

1.1.2.7 przetwarzaj_polyline()

```
main.przetwarzaj_polyline (
                element )
```

Definition at line 123 of file main.py.

```
00123 def przetwarzaj_polyline(element):
00124
00127     if hasattr(element, 'points') and element.points:
00128         return list(element.points)
00129     return []
00130
```

1.1.2.8 przetwarzaj_rect()

```
main.przetwarzaj_rect (
              element )
```

Definition at line 47 of file main.py.

```
00047 def przetwarzaj_rect(element):
00048
00054     x = element.x or 0
00055     y = element.y or 0
00056     w = element.width or 0
00057     h = element.height or 0
00058
00059     # Punkty narożników prostokąta: lewy-górny -> prawy-górny -> prawy-dolny -> lewy-dolny ->
      zamknięcie
00060     punkty = [
00061         (x, y),
00062         (x + w, y),
00063         (x + w, y + h),
00064         (x, y + h),
00065         (x, y)  # zamknięcie
00066     ]
00067     return punkty
00068
```

1.1.3 Variable Documentation

1.1.3.1 gcode_wyjsciowy

```
list main.gcode_wyjsciowy = []
```

Lista poleceń G-code do zapisania do pliku.

Definition at line 28 of file main.py.

1.1.3.2 punkty

```
main.punkty = []
```

Definition at line 166 of file main.py.

1.1.3.3 Rozdzielczosc

```
int main.Rozdzielczosc = 1
```

Rozdzielczość próbkowania - maksymalna odległość między punktami.

Definition at line 19 of file main.py.

1.1.3.4 skala

```
float main.skala = 1.0
```

Skala konwersji współrzędnych SVG na jednostki G-code.

Definition at line 16 of file main.py.

1.1.3.5 svg

```
main.svg = SVG.parse("output_path_arial.svg")
```

Wczytana zawartość pliku SVG.

Definition at line 25 of file main.py.

Chapter 2

File Documentation

2.1 main.py File Reference

Namespaces

- namespace main

Functions

- main.probkowanie (path, rozdzielczosc)
- main.przetwarzaj_rect (element)
- main.przetwarzaj_circle (element)
- main.przetwarzaj_ellipse (element)
- main.przetwarzaj_line (element)
- main.przetwarzaj_polyline (element)
- main.przetwarzaj_polygon (element)
- main.generuj_gcode_dla_punktow (punkty)

Variables

- float main.skala = 1.0

  Skala konwersji współrzędnych SVG na jednostki G-code.
- int main.Rozdzielczosc = 1

  Rozdzielczość próbkowania - maksymalna odległość między punktami.
- main.svg = SVG.parse("output_path_arial.svg")

  Wczytana zawartość pliku SVG.
- list main.gcode_wyjsciowy = [ ]

  Lista poleceń G-code do zapisania do pliku.
- list main.punkty = [ ]

2.2 main.py

Go to the documentation of this file.

```
00001 """
00002 @file main.py
00003 @brief Główny moduł konwersji SVG do G-code
00004 @details Konwertuje elementy SVG (prostokąty, okręgi, elipsy, linie, ścieżki)
00005          na sekwencje poleceń G-code dla urządzeń CNC/maszyn rysujących.
00006 @author RoboScribe
00007 @version 1.0
00008 """
00009
00010 from svgelements import SVG, Rect, Circle, Ellipse, Line, SimpleLine, Polyline, Polygon, Text
00011 import numpy as np
00012 import math
00013
00014 # ---- konfiguracja ----
00015
00016 skala = 1.0
00017
00018
00019 Rozdzielczosc = 1
00020
00021
00022
00023 # ---- wczytanie SVG ----
00024
00025 svg = SVG.parse("output_path_arial.svg")
00026
00027
00028 gcode_wyjsciowy = []
00029
00030 def probkowanie(path, rozdzielczosc):
00031
00035     punkty = []
00036     dlugosc_sciezki = path.length()
00037     #TODO: rozważyć jak ma działać rozdzielczość
00038     #n = max(1, int(dlugosc_sciezki / rozdzielczosc))
00039     n = int(dlugosc_sciezki / rozdzielczosc)
00040
00041     for i in range(n+1):
00042         p = path.point(i / n)
00043         punkty.append((p.real, p.imag))
00044     return punkty
00045
00046
00047 def przetwarzaj_rect(element):
00048
00054     x = element.x or 0
00055     y = element.y or 0
00056     w = element.width or 0
00057     h = element.height or 0
00058
00059     # Punkty narożników prostokąta: lewy-górny -> prawy-górny -> prawy-dolny -> lewy-dolny ->
      zamknięcie
00060     punkty = [
00061         (x, y),
00062         (x + w, y),
00063         (x + w, y + h),
00064         (x, y + h),
00065         (x, y)  # zamknięcie
00066     ]
00067     return punkty
00068
00069 def przetwarzaj_circle(element):
00070
00074     cx = element.cx or 0
00075     cy = element.cy or 0
00076     r = element.r or 0
00077
00078     # Liczba segmentów na podstawie Rozdzielczości
00079     segments = max(int(2 * math.pi * r / Rozdzielczosc), 8)
00080
00081     punkty = []
00082     for i in range(segments + 1):
00083         angle = (i / segments) * 2 * math.pi
00084         x = cx + r * math.cos(angle)
00085         y = cy + r * math.sin(angle)
00086         punkty.append((x, y))
00087
00088     return punkty
00089
00090 def przetwarzaj_ellipse(element):
00091
00095     cx = element.cx or 0
```

```
00096     cy = element.cy or 0
00097     rx = element.rx or 0
00098     ry = element.ry or 0
00099 
00100     # Liczba segmentów na podstawie średniej Rozdzielczości
00101     segments = max(int(2 * math.pi * max(rx, ry) / Rozdzielczosc), 8)
00102 
00103     punkty = []
00104     for i in range(segments + 1):
00105         angle = (i / segments) * 2 * math.pi
00106         x = cx + rx * math.cos(angle)
00107         y = cy + ry * math.sin(angle)
00108         punkty.append((x, y))
00109 
00110     return punkty
00111 
00112 def przetwarzaj_line(element):
00113 
00116     x1 = element.x1 or 0
00117     y1 = element.y1 or 0
00118     x2 = element.x2 or 0
00119     y2 = element.y2 or 0
00120 
00121     return [(x1, y1), (x2, y2)]
00122 
00123 def przetwarzaj_polyline(element):
00124 
00127     if hasattr(element, 'points') and element.points:
00128         return list(element.points)
00129     return []
00130 
00131 def przetwarzaj_polygon(element):
00132 
00136     if hasattr(element, 'points') and element.points:
00137         punkty = list(element.points)
00138         if punkty and punkty[0] != punkty[-1]:
00139             punkty.append(punkty[0])  # zamknięcie
00140         return punkty
00141     return []
00142 
00143 def generuj_gcode_dla_punktow(punkty):
00144 
00148     if not punkty:
00149         return
00150 
00151     # Przejście do pierwszego punktu bez rysowania
00152     x0, y0 = punkty[0]
00153     x_val = round(x0 * skala, 2)
00154     y_val = round(y0 * skala, 2)
00155     gcode_wyjsciowy.append(f"G0 X{x_val} Y{y_val}")
00156 
00157     gcode_wyjsciowy.append('G0 Z0')  # Opuszczenie pisaka
00158 
00159     # Rysowanie pozostałych punktów
00160     for x, y in punkty[1:]:
00161         x_val = round(x * skala, 2)
00162         y_val = round(y * skala, 2)
00163         gcode_wyjsciowy.append(f"G1 X{x_val} Y{y_val}")
00164 
00165 for element in svg.elements():
00166     punkty = []
00167 
00168     if hasattr(element, "d"):  # element ma ścieżkę (Path)
00169         punkty = probkowanie(element, Rozdzielczosc)
00170     elif isinstance(element, Rect):
00171         punkty = przetwarzaj_rect(element)
00172     elif isinstance(element, Circle):
00173         punkty = przetwarzaj_circle(element)
00174     elif isinstance(element, Ellipse):
00175         punkty = przetwarzaj_ellipse(element)
00176     elif isinstance(element, (Line, SimpleLine)):
00177         punkty = przetwarzaj_line(element)
00178     elif isinstance(element, Polyline):
00179         punkty = przetwarzaj_polyline(element)
00180     elif isinstance(element, Polygon):
00181         punkty = przetwarzaj_polygon(element)
00182     elif isinstance(element, Text):
00183         print(f"Nieobsługiwany element: {type(element)} - przekonwertuj tekst na ścieżkę w edytorze
      SVG.")
00184     else:
00185         print(f"Nieobsługiwany element: {type(element)}")
00186         continue
00187 
00188     generuj_gcode_dla_punktow(punkty)
00189     if punkty:
00190         gcode_wyjsciowy.append("G0 Z1")  # Podniesienie pisaka po zakończeniu kształtu
00191 
```

```
00192 #TODO: zamiana współrzędnych X Y na kąty dla serwomechanizmów rysujących
00193
00194 # ---- zapis ----
00195 with open("rysunek.gcode", "w") as f:
00196     f.write("\n".join(gcode_wyjsciowy))
00197
00198 print("Gotowe! Wygenerowano plik rysunek.gcode")
```

Index

gcode_wyjsciowy
    main, 4
generuj_gcode_dla_punktow
    main, 2

main, 1
    gcode_wyjsciowy, 4
    generuj_gcode_dla_punktow, 2
    probkowanie, 2
    przetwarzaj_circle, 2
    przetwarzaj_ellipse, 2
    przetwarzaj_line, 3
    przetwarzaj_polygon, 3
    przetwarzaj_polyline, 3
    przetwarzaj_rect, 3
    punkty, 4
    Rozdzielczosc, 4
    skala, 4
    svg, 4
main.py, 5

probkowanie
    main, 2
przetwarzaj_circle
    main, 2
przetwarzaj_ellipse
    main, 2
przetwarzaj_line
    main, 3
przetwarzaj_polygon
    main, 3
przetwarzaj_polyline
    main, 3
przetwarzaj_rect
    main, 3
punkty
    main, 4

Rozdzielczosc
    main, 4

skala
    main, 4
svg
    main, 4