svg-reader

0.3

Generated by Doxygen 1.9.1

1 Hierarchical Index 1

1.1 Class Hierarchy . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 Class Documentation 5

3.1 Circle Class Reference . . . 5

3.1.1 Detailed Description . . . 6

3.1.2 Constructor & Destructor Documentation . . . 6

3.1.2.1 Circle() . . . 7

3.1.3 Member Function Documentation . . . 7

3.1.3.1 getClass() . . . 7

3.2 ColorShape Class Reference . . . 8

3.2.1 Detailed Description . . . 9

3.2.2 Constructor & Destructor Documentation . . . 9

3.2.2.1 ColorShape() [1/3] . . . 9

3.2.2.2 ColorShape() [2/3] . . . 9

3.2.2.3 ColorShape() [3/3] . . . 10

3.2.3 Friends And Related Function Documentation . . . 10

3.2.3.1 operator<< . . . 10

3.3 Ell Class Reference . . . 11

3.3.1 Detailed Description . . . 12

3.3.2 Constructor & Destructor Documentation . . . 13

3.3.2.1 Ell() . . . 13

3.3.3 Member Function Documentation . . . 13

3.3.3.1 getClass() . . . 13

3.3.3.2 getMaxBound() . . . 14

3.3.3.3 getMinBound() . . . 14

3.3.3.4 getRadius() . . . 14

3.3.3.5 printData() . . . 15

3.3.3.6 setRadius() . . . 15

3.4 Gradient Class Reference . . . 15

3.4.1 Detailed Description . . . 17

3.4.2 Constructor & Destructor Documentation . . . 17

3.4.2.1 Gradient() . . . 17

3.4.3 Member Function Documentation . . . 17

3.4.3.1 addStop() . . . 17

3.4.3.2 getClass() . . . 18

3.4.3.3 getPoints() . . . 18

3.4.3.4 getStops() . . . 18

3.4.3.5 getTransforms() . . . 19

3.4.3.6 getUnits() . . . 19

3.4.3.7 setTransforms() . . . 19
3.4.3.8 setUnits() . . . 20
3.5 Group Class Reference . . . 20
3.5.1 Detailed Description . . . 22
3.5.2 Constructor & Destructor Documentation . . . 22
3.5.2.1 Group() . . . 22
3.5.3 Member Function Documentation . . . 22
3.5.3.1 addElement() . . . 22
3.5.3.2 getAttributes() . . . 23
3.5.3.3 getClass() . . . 23
3.5.3.4 getElements() . . . 23
3.5.3.5 printData() . . . 24
3.6 Line Class Reference . . . 24
3.6.1 Detailed Description . . . 25
3.6.2 Constructor & Destructor Documentation . . . 26
3.6.2.1 Line() . . . 26
3.6.3 Member Function Documentation . . . 26
3.6.3.1 getClass() . . . 26
3.6.3.2 getDirection() . . . 27
3.6.3.3 getLength() . . . 27
3.6.3.4 setDirection() . . . 27
3.7 LinearGradient Class Reference . . . 28
3.7.1 Detailed Description . . . 28
3.7.2 Constructor & Destructor Documentation . . . 29
3.7.2.1 LinearGradient() . . . 29
3.7.3 Member Function Documentation . . . 29
3.7.3.1 getClass() . . . 29
3.8 Parser Class Reference . . . 30
3.8.1 Detailed Description . . . 32
3.8.2 Constructor & Destructor Documentation . . . 32
3.8.2.1 Parser() . . . 32
3.8.3 Member Function Documentation . . . 32
3.8.3.1 getAttribute() . . . 33
3.8.3.2 getFloatAttribute() . . . 33
3.8.3.3 GetGradients() . . . 34
3.8.3.4 getGradientStops() . . . 35
3.8.3.5 getInstance() . . . 36
3.8.3.6 getRoot() . . . 36
3.8.3.7 getTransformOrder() . . . 36
3.8.3.8 getViewBox() . . . 37
3.8.3.9 getViewPort() . . . 37
3.8.3.10 parseCircle() . . . 38

3.8.3.11 parseColor() . . . 38
3.8.3.12 parseElements() . . . 39
3.8.3.13 parseEllipse() . . . 41
3.8.3.14 parseGradient() . . . 42
3.8.3.15 parseLine() . . . 42
3.8.3.16 parsePath() . . . 43
3.8.3.17 parsePathPoints() . . . 43
3.8.3.18 parsePoints() . . . 45
3.8.3.19 parsePolygon() . . . 46
3.8.3.20 parsePolyline() . . . 47
3.8.3.21 parseRect() . . . 47
3.8.3.22 parseShape() . . . 48
3.8.3.23 parseText() . . . 49
3.8.3.24 printShapesData() . . . 49
3.8.4 Member Data Documentation . . . 50
3.8.4.1 gradients . . . 50
3.9 Path Class Reference . . . 50
3.9.1 Detailed Description . . . 52
3.9.2 Constructor & Destructor Documentation . . . 52
3.9.2.1 Path() . . . 52
3.9.3 Member Function Documentation . . . 52
3.9.3.1 addPoint() . . . 52
3.9.3.2 getClass() . . . 53
3.9.3.3 getFillRule() . . . 53
3.9.3.4 getPoints() . . . 54
3.9.3.5 printData() . . . 54
3.9.3.6 setFillRule() . . . 54
3.10 PathPoint Struct Reference . . . 55
3.10.1 Detailed Description . . . 56
3.11 Plygon Class Reference . . . 56
3.11.1 Detailed Description . . . 57
3.11.2 Constructor & Destructor Documentation . . . 57
3.11.2.1 Plygon() . . . 58
3.11.3 Member Function Documentation . . . 59
3.11.3.1 getClass() . . . 59
3.12 Plyline Class Reference . . . 59
3.12.1 Detailed Description . . . 61
3.12.2 Constructor & Destructor Documentation . . . 61
3.12.2.1 Plyline() . . . 61
3.12.3 Member Function Documentation . . . 61
3.12.3.1 getClass() . . . 61
3.13 PolyShape Class Reference . . . 62

3.13.1 Detailed Description . . . 64
3.13.2 Constructor & Destructor Documentation . . . 64
3.13.2.1 PolyShape() . . . 64
3.13.3 Member Function Documentation . . . 64
3.13.3.1 addPoint() . . . 64
3.13.3.2 getClass() . . . 65
3.13.3.3 getFillRule() . . . 65
3.13.3.4 getMaxBound() . . . 65
3.13.3.5 getMinBound() . . . 66
3.13.3.6 getPoints() . . . 66
3.13.3.7 printData() . . . 66
3.13.3.8 setFillRule() . . . 66
3.14 RadialGradient Class Reference . . . 67
3.14.1 Detailed Description . . . 68
3.14.2 Constructor & Destructor Documentation . . . 68
3.14.2.1 RadialGradient() . . . 68
3.14.3 Member Function Documentation . . . 69
3.14.3.1 getClass() . . . 69
3.14.3.2 getRadius() . . . 69
3.15 Rect Class Reference . . . 70
3.15.1 Detailed Description . . . 71
3.15.2 Constructor & Destructor Documentation . . . 71
3.15.2.1 Rect() . . . 71
3.15.3 Member Function Documentation . . . 72
3.15.3.1 getClass() . . . 72
3.15.3.2 getHeight() . . . 72
3.15.3.3 getRadius() . . . 73
3.15.3.4 getWidth() . . . 73
3.15.3.5 printData() . . . 73
3.15.3.6 setHeight() . . . 73
3.15.3.7 setRadius() . . . 74
3.15.3.8 setWidth() . . . 74
3.16 Renderer Class Reference . . . 74
3.16.1 Detailed Description . . . 76
3.16.2 Member Function Documentation . . . 76
3.16.2.1 applyTransform() . . . 76
3.16.2.2 applyTransformsOnBrush() [1/2] . . . 77
3.16.2.3 applyTransformsOnBrush() [2/2] . . . 78
3.16.2.4 draw() . . . 78
3.16.2.5 drawCircle() . . . 79
3.16.2.6 drawEllipse() . . . 80
3.16.2.7 drawLine() . . . 81

3.16.2.8 drawPath() . . . 81
3.16.2.9 drawPolygon() . . . 84
3.16.2.10 drawPolyline() . . . 85
3.16.2.11 drawRectangle() . . . 86
3.16.2.12 drawText() . . . 87
3.16.2.13 getBrush() . . . 88
3.16.2.14 getInstance() . . . 90
3.17 Stop Class Reference . . . 90
3.17.1 Detailed Description . . . 91
3.17.2 Constructor & Destructor Documentation . . . 91
3.17.2.1 Stop() . . . 92
3.17.3 Member Function Documentation . . . 92
3.17.3.1 getColor() . . . 92
3.17.3.2 getOffset() . . . 92
3.18 SVGElement Class Reference . . . 93
3.18.1 Detailed Description . . . 95
3.18.2 Constructor & Destructor Documentation . . . 95
3.18.2.1 SVGElement() [1/3] . . . 95
3.18.2.2 SVGElement() [2/3] . . . 95
3.18.2.3 SVGElement() [3/3] . . . 96
3.18.3 Member Function Documentation . . . 96
3.18.3.1 addElement() . . . 96
3.18.3.2 getClass() . . . 97
3.18.3.3 getFillColor() . . . 97
3.18.3.4 getGradient() . . . 98
3.18.3.5 getMaxBound() . . . 98
3.18.3.6 getMinBound() . . . 98
3.18.3.7 getOutlineColor() . . . 99
3.18.3.8 getOutlineThickness() . . . 99
3.18.3.9 getParent() . . . 99
3.18.3.10 getPosition() . . . 100
3.18.3.11 getTransforms() . . . 100
3.18.3.12 printData() . . . 101
3.18.3.13 setFillColor() . . . 101
3.18.3.14 setGradient() . . . 101
3.18.3.15 setOutlineColor() . . . 102
3.18.3.16 setOutlineThickness() . . . 102
3.18.3.17 setParent() . . . 103
3.18.3.18 setPosition() [1/2] . . . 103
3.18.3.19 setPosition() [2/2] . . . 104
3.18.3.20 setTransforms() . . . 104
3.19 Text Class Reference . . . 105

3.19.1 Detailed Description . . . 106
3.19.2 Constructor & Destructor Documentation . . . 106
3.19.2.1 Text() . . . 107
3.19.3 Member Function Documentation . . . 107
3.19.3.1 getAnchor() . . . 107
3.19.3.2 getClass() . . . 107
3.19.3.3 getContent() . . . 108
3.19.3.4 getFontSize() . . . 108
3.19.3.5 getFontStyle() . . . 108
3.19.3.6 setAnchor() . . . 108
3.19.3.7 setContent() . . . 109
3.19.3.8 setFontSize() . . . 109
3.19.3.9 setFontStyle() . . . 109
3.20 Vector2D< T > Class Template Reference . . . 110
3.20.1 Detailed Description . . . 110
3.20.2 Constructor & Destructor Documentation . . . 111
3.20.2.1 Vector2D() [1/3] . . . 111
3.20.2.2 Vector2D() [2/3] . . . 111
3.20.2.3 Vector2D() [3/3] . . . 111
3.21 ViewBox Class Reference . . . 112
3.21.1 Detailed Description . . . 112
3.21.2 Constructor & Destructor Documentation . . . 112
3.21.2.1 ViewBox() [1/2] . . . 113
3.21.2.2 ViewBox() [2/2] . . . 113
3.21.3 Member Function Documentation . . . 113
3.21.3.1 getHeight() . . . 113
3.21.3.2 getWidth() . . . 114
3.21.3.3 getX() . . . 114
3.21.3.4 getY() . . . 114
3.22 Viewer Class Reference . . . 115
3.22.1 Detailed Description . . . 116
3.22.2 Member Function Documentation . . . 116
3.22.2.1 getInstance() . . . 117
3.22.2.2 getWindowSize() . . . 117
3.22.2.3 handleKeyDown() . . . 117
3.22.2.4 handleKeyEvent() . . . 118
3.22.2.5 handleLeftButtonDown() . . . 118
3.22.2.6 handleMouseEvent() . . . 118
3.22.2.7 handleMouseMove() . . . 119
3.22.2.8 handleMouseWheel() . . . 120
3.22.3 Member Data Documentation . . . 120
3.22.3.1 needs_repaint . . . 120

Index 121

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ColorShape . . . 8
Gradient . . . 15
  LinearGradient . . . 28
  RadialGradient . . . 67
Parser . . . 30
PathPoint . . . 55
Renderer . . . 74
Stop . . . 90
SVGElement . . . 93
  Ell . . . 11
    Circle . . . 5
  Group . . . 20
  Line . . . 24
  Path . . . 50
  PolyShape . . . 62
    Plygon . . . 56
    Plyline . . . 59
  Rect . . . 70
  Text . . . 105
Vector2D< T > . . . 110
Vector2D< float > . . . 110
ViewBox . . . 112
Viewer . . . 115

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Circle
: Represents a circle in 2D space . . . 5

ColorShape
: Utility class for manipulating RGBA ColorShapes . . . 8

Ell
: Represents an ellipse in 2D space . . . 11

Gradient
: A class that represents a gradient . . . 15

Group
: A composite class that contains a vector of shape pointers (polymorphic) . . . 20

Line
: Represents a line in 2D space . . . 24

LinearGradient
: A class that represents a linear gradient . . . 28

Parser
: To manipulate and parse an SVG file . . . 30

Path
: Represents a path element in 2D space . . . 50

PathPoint
: A struct that contains a point and a type of point . . . 55

Plygon
: Represents a polygon in 2D space . . . 56

Plyline
: Represents a polyline in 2D space . . . 59

PolyShape
: Abstract base class for polygon and polyline shapes in 2D space . . . 62

RadialGradient
: A class that represents a radial gradient . . . 67

Rect
: Represents a rectangle in 2D space . . . 70

Renderer
: Singleton class responsible for rendering shapes using GDI+ . . . 74

Stop
: A class that represents a stop . . . 90

SVGElement
: Represents an element in an SVG file . . . 93

Text
Represents text in 2D space 105
Vector2D< T >
Utility template class for manipulating 2-dimensional vectors 110
ViewBox
A ViewBox is a rectangle that defines the area of the SVG canvas that should be visible to the user 112
Viewer
Represents a viewer for rendering and interacting with a scene 115

Chapter 3

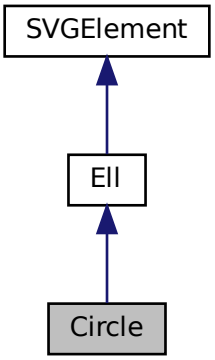
Class Documentation

3.1 Circle Class Reference

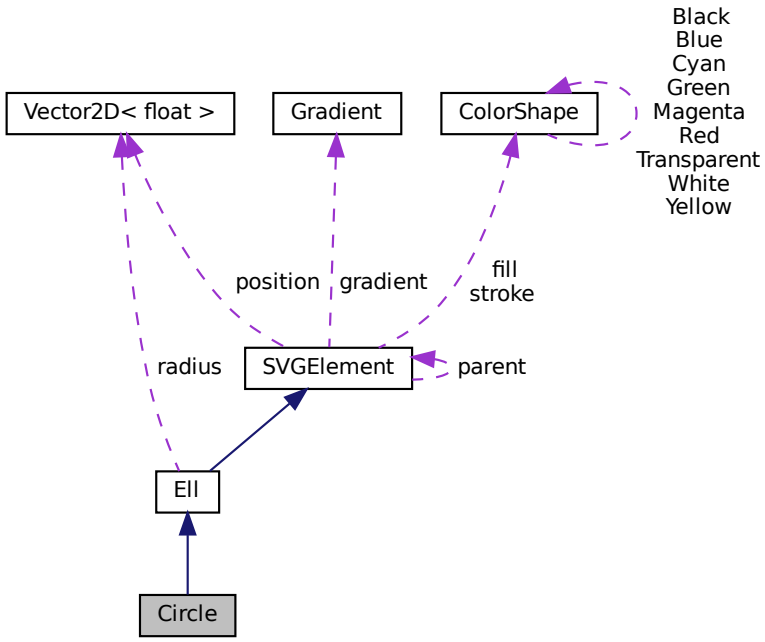
Represents a circle in 2D space.

`#include <Circle.hpp>`

Inheritance diagram for Circle:



Collaboration diagram for Circle:



Public Member Functions

- Circle (float radius, const Vector2Df &center, ColorShape fill, ColorShape stroke, float stroke_width)

  Constructs a Circle object.
- std::string getClass () const override

  Gets the type of the shape.

Additional Inherited Members

3.1.1 Detailed Description

Represents a circle in 2D space.

The Circle class is derived from the Ellipse class and defines a circle with a specified radius, center, fill color, stroke color, and stroke thickness.

Definition at line 13 of file Circle.hpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Circle()

```
Circle::Circle (
            float radius,
            const Vector2Df & center,
            ColorShape fill,
            ColorShape stroke,
            float stroke_width )
```

Constructs a Circle object.

Parameters

radius	The radius of the circle.
center	The center of the circle.
fill	Fill color of the circle.
stroke	Outline color of the circle.
stroke_width	Thickness of the circle outline.

Definition at line 3 of file Circle.cpp.

```
5     : Ell(Vector2Df(radius, radius), center, fill, stroke, stroke_width) {}
```

3.1.3 Member Function Documentation

3.1.3.1 getClass()

```
std::string Circle::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Circle".

Implements SVGElement.

Definition at line 7 of file Circle.cpp.

```
7 { return "Circle"; }
```

The documentation for this class was generated from the following files:

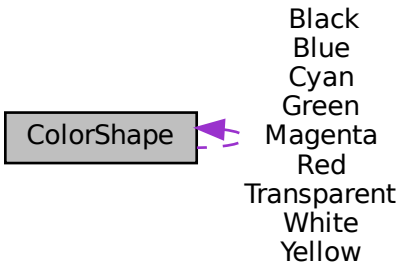
- src/graphics/Circle.hpp
- src/graphics/Circle.cpp

3.2 ColorShape Class Reference

Utility class for manipulating RGBA ColorShapes.

`#include <ColorShape.hpp>`

Collaboration diagram for ColorShape:



Public Member Functions

- ColorShape ()

  Default constructor.
- ColorShape (int red, int green, int blue, int alpha=255)

  Construct the ColorShape from its 4 RGBA components.
- ColorShape (int color)

  Construct the color from 32-bit unsigned integer.

Public Attributes

- int r

  Red component.
- int g

  Green component.
- int b

  Blue component.
- int a

  Alpha (opacity) component.

Static Public Attributes

- static const ColorShape Black
  Black predefined color.
- static const ColorShape White
  White predefined color.
- static const ColorShape Red
  Red predefined color.
- static const ColorShape Green
  Green predefined color.
- static const ColorShape Blue
  Blue predefined color.
- static const ColorShape Yellow
  Yellow predefined color.
- static const ColorShape Magenta
  Magenta predefined color.
- static const ColorShape Cyan
  Cyan predefined color.
- static const ColorShape Transparent
  Transparent (black) predefined color.

Friends

- std::ostream & operator<< (std::ostream &os, const ColorShape &color)
  Prints the color.

3.2.1 Detailed Description

Utility class for manipulating RGBA ColorShapes.

Definition at line 11 of file ColorShape.hpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ColorShape() [1/3]

```
ColorShape::ColorShape ( )
```

Default constructor.

Constructs an opaque black ColorShape. It is equivalent to ColorShape(0, 0, 0, 255).

Definition at line 14 of file ColorShape.cpp.

```
14 : r(0), g(0), b(0), a(255) {}
```

3.2.2.2 ColorShape() [2/3]

```
ColorShape::ColorShape (
            int red,
            int green,
            int blue,
            int alpha = 255 )
```

Construct the ColorShape from its 4 RGBA components.

Parameters

red	Red component (in the range [0, 255])
green	Green component (in the range [0, 255])
blue	Blue component (in the range [0, 255])
alpha	Alpha (opacity) component (in the range [0, 255])

Definition at line 16 of file ColorShape.cpp.

```
17     : r(red), g(green), b(blue), a(alpha) {
18     r = std::clamp(r, 0, 255);
19     g = std::clamp(g, 0, 255);
20     b = std::clamp(b, 0, 255);
21     a = std::clamp(a, 0, 255);
22 }
```

3.2.2.3 ColorShape() [3/3]

```
ColorShape::ColorShape (
            int color )  [explicit]
```

Construct the color from 32-bit unsigned integer.

Parameters

color	Number containing the RGBA components (in that order)

Definition at line 24 of file ColorShape.cpp.

```
25     : r(static_cast< int >((color & 0xff000000) » 24)),
26       g(static_cast< int >((color & 0x00ff0000) » 16)),
27       b((color & 0x0000ff00) » 8), a((color & 0x000000ff) » 0) {}
```

3.2.3 Friends And Related Function Documentation

3.2.3.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const ColorShape & color )  [friend]
```

Prints the color.

Parameters

os	output stream
color	color to be printed

Returns

output stream

Note

This function is used for printing the color.

Definition at line 29 of file ColorShape.cpp.

```
29                                                          {
30     os « "Color(" « color.r « ", " « color.g « ", " « color.b « ", "
31        « color.a « ")";
32     return os;
33 }
```

The documentation for this class was generated from the following files:

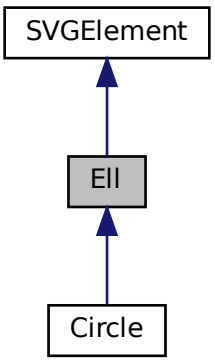
- src/graphics/ColorShape.hpp
- src/graphics/ColorShape.cpp

3.3 Ell Class Reference

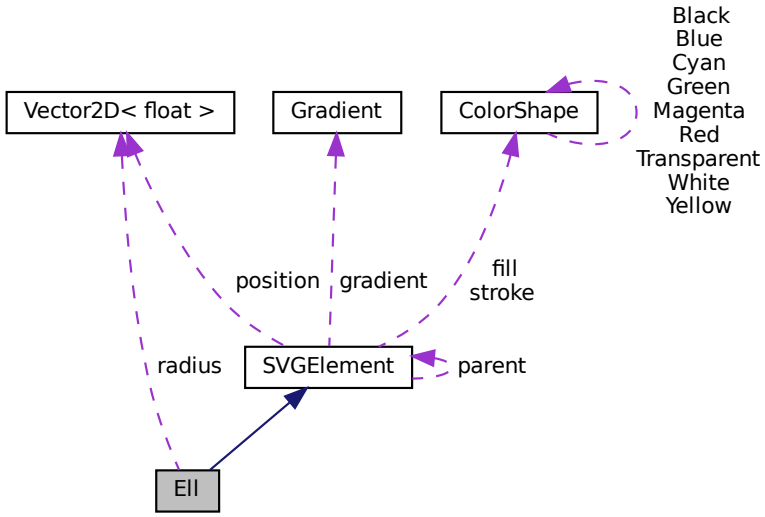
Represents an ellipse in 2D space.

`#include <Ellipse.hpp>`

Inheritance diagram for Ell:



Collaboration diagram for Ell:



Public Member Functions

- Ell (const Vector2Df &radius, const Vector2Df &center, ColorShape fill, ColorShape stroke, float stroke_width)
  Constructs an Ellipse object.
- std::string getClass () const override
  Gets the type of the shape.
- void setRadius (const Vector2Df &radius)
  Sets the radius of the ellipse.
- Vector2Df getRadius () const
  Gets the radius of the ellipse.
- Vector2Df getMinBound () const override
  Gets the minimum bounding box of the shape.
- Vector2Df getMaxBound () const override
  Gets the maximum bounding box of the shape.
- void printData () const override
  Prints the data of the shape.

Private Attributes

- Vector2Df radius
  Radii of the ellipse in the x and y directions.

Additional Inherited Members

3.3.1 Detailed Description

Represents an ellipse in 2D space.

The Ellipse class is derived from the SVGElement class and defines an ellipse with a variable radius in the x and y directions.

Definition at line 12 of file Ellipse.hpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Ell()

```
Ell::Ell (
            const Vector2Df & radius,
            const Vector2Df & center,
            ColorShape fill,
            ColorShape stroke,
            float stroke_width )
```

Constructs an Ellipse object.

Parameters

radius	The radii of the ellipse in the x and y directions.
center	The center of the ellipse.
fill	Fill color of the ellipse.
stroke	Outline color of the ellipse.
stroke_width	Thickness of the ellipse outline.

Definition at line 5 of file Ellipse.cpp.

```
7     : SVGElement(fill, stroke, stroke_thickness, center), radius(radius) {}
```

3.3.3 Member Function Documentation

3.3.3.1 getClass()

```
std::string Ell::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Ellipse".

Note

This function is used for determining the type of the shape.

Implements SVGElement.

Definition at line 9 of file Ellipse.cpp.

```
9 { return "Ellipse"; }
```

3.3.3.2 getMaxBound()

```
Vector2Df Ell::getMaxBound ( ) const  [override], [virtual]
```

Gets the maximum bounding box of the shape.

Returns

The maximum bounding box of the shape.

Reimplemented from SVGElement.

Definition at line 20 of file Ellipse.cpp.

```
20                                      {
21     return Vector2Df(getPosition().x + getRadius().x,
22                      getPosition().y + getRadius().y);
23 }
```

3.3.3.3 getMinBound()

```
Vector2Df Ell::getMinBound ( ) const  [override], [virtual]
```

Gets the minimum bounding box of the shape.

Returns

The minimum bounding box of the shape.

Reimplemented from SVGElement.

Definition at line 15 of file Ellipse.cpp.

```
15                                      {
16     return Vector2Df(getPosition().x - getRadius().x,
17                      getPosition().y - getRadius().y);
18 }
```

3.3.3.4 getRadius()

```
Vector2Df Ell::getRadius ( ) const
```

Gets the radius of the ellipse.

Returns

The radius of the ellipse.

Definition at line 13 of file Ellipse.cpp.

```
13 { return radius; }
```

3.3.3.5 printData()

```
void Ell::printData ( ) const  [override], [virtual]
```

Prints the data of the shape.

Note

This function is used for debugging purposes.

Reimplemented from SVGElement.

Definition at line 25 of file Ellipse.cpp.

```
25                              {
26     SVGElement::printData();
27     std::cout « "Radius: " « getRadius().x « " " « getRadius().y
28               « std::endl;
29 }
```

3.3.3.6 setRadius()

```
void Ell::setRadius (
             const Vector2Df & radius )
```

Sets the radius of the ellipse.

Parameters

radius	The new radius of the ellipse.

Definition at line 11 of file Ellipse.cpp.

```
11 { this->radius = radius; }
```

The documentation for this class was generated from the following files:

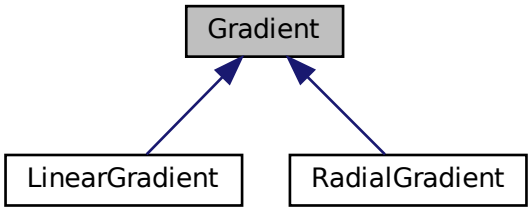
- src/graphics/Ellipse.hpp
- src/graphics/Ellipse.cpp

3.4 Gradient Class Reference

A class that represents a gradient.

```
#include <Gradient.hpp>
```

Inheritance diagram for Gradient:



Public Member Functions

- Gradient (std::vector< Stop > stops, std::pair< Vector2Df, Vector2Df > points, std::string units)

  Constructs a Gradient object.
- virtual ~Gradient ()=default

  Destructs a Gradient object.
- virtual std::string getClass () const =0

  Gets the type of the gradient.
- std::vector< Stop > getStops () const

  Gets the stops of the gradient.
- std::pair< Vector2Df, Vector2Df > getPoints () const

  Gets the start and end points of the gradient.
- void setUnits (std::string units)

  Gets the units of the gradient.
- std::string getUnits () const

  Gets the units of the gradient.
- void setTransforms (std::vector< std::string > transforms)

  Gets the transforms of the gradient.
- std::vector< std::string > getTransforms () const

  Gets the transforms of the gradient.
- void addStop (Stop stop)

  Adds a stop to the gradient.

Private Attributes

- std::vector< Stop > stops

  Stops of the gradient.
- std::pair< Vector2Df, Vector2Df > points

  Start and end points of the gradient.
- std::string units

  Units of the gradient.
- std::vector< std::string > transforms

  Transforms of the gradient.

3.4.1 Detailed Description

A class that represents a gradient.

The Gradient class is an abstract class that represents a gradient. It contains a vector of Stop objects that represent the stops of the gradient. It also contains a pair of Vector2D objects that represent the start and end points of the gradient.

Definition at line 18 of file Gradient.hpp.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Gradient()

```
Gradient::Gradient (
            std::vector< Stop > stops,
            std::pair< Vector2Df, Vector2Df > points,
            std::string units )
```

Constructs a Gradient object.

Parameters

stops	The stops of the gradient.
points	The start and end points of the gradient.
units	The units of the gradient.

Definition at line 3 of file Gradient.cpp.

```
5     : stops(stops), points(points), units(units) {}
```

3.4.3 Member Function Documentation

3.4.3.1 addStop()

```
void Gradient::addStop (
            Stop stop )
```

Adds a stop to the gradient.

Parameters

stop	The stop to be added to the gradient.

Definition at line 23 of file Gradient.cpp.
```
23 { stops.push_back(stop); }
```

3.4.3.2 getClass()

```
virtual std::string Gradient::getClass ( ) const  [pure virtual]
```

Gets the type of the gradient.

Returns

The string that represents the type of the gradient.

Implemented in RadialGradient, and LinearGradient.

3.4.3.3 getPoints()

```
std::pair< Vector2Df, Vector2Df > Gradient::getPoints ( ) const
```

Gets the start and end points of the gradient.

Returns

The start and end points of the gradient.

Definition at line 9 of file Gradient.cpp.
```
9 { return points; }
```

3.4.3.4 getStops()

```
std::vector< Stop > Gradient::getStops ( ) const
```

Gets the stops of the gradient.

Returns

The stops of the gradient.

Definition at line 7 of file Gradient.cpp.
```
7 { return stops; }
```

3.4.3.5 getTransforms()

```
std::vector< std::string > Gradient::getTransforms ( ) const
```

Gets the transforms of the gradient.

Returns

The transforms of the gradient.

Definition at line 19 of file Gradient.cpp.

```
19                                                      {
20     return transforms;
21 }
```

3.4.3.6 getUnits()

```
std::string Gradient::getUnits ( ) const
```

Gets the units of the gradient.

Returns

The units of the gradient.

Definition at line 13 of file Gradient.cpp.

```
13 { return units; }
```

3.4.3.7 setTransforms()

```
void Gradient::setTransforms (
            std::vector< std::string > transforms )
```

Gets the transforms of the gradient.

Returns

The transforms of the gradient.

Definition at line 15 of file Gradient.cpp.

```
15                                                      {
16     this->transforms = transforms;
17 }
```

3.4.3.8 setUnits()

```
void Gradient::setUnits (
            std::string units )
```

Gets the units of the gradient.

Returns

The units of the gradient.

Definition at line 11 of file Gradient.cpp.

```
11 { this->units = units; }
```

The documentation for this class was generated from the following files:

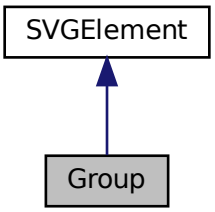
- src/graphics/Gradient.hpp
- src/graphics/Gradient.cpp

3.5 Group Class Reference

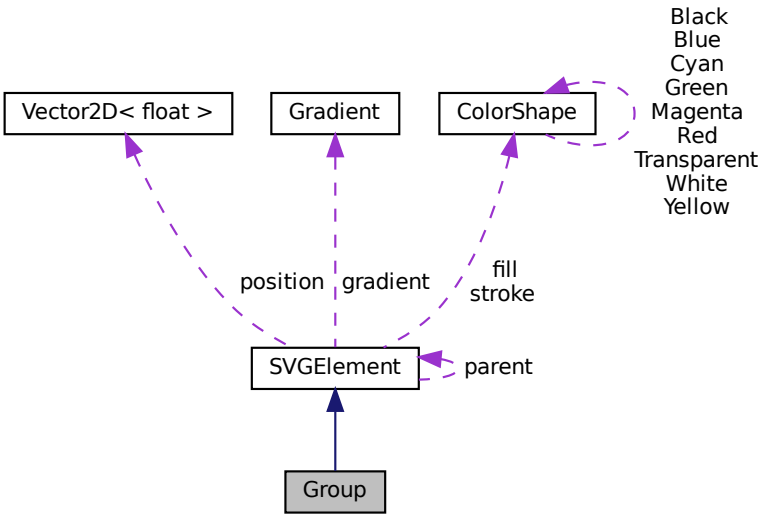
A composite class that contains a vector of shape pointers (polymorphic).

```
#include <Group.hpp>
```

Inheritance diagram for Group:



Collaboration diagram for Group:



Public Member Functions

- Group ()

  Constructs a Group object.
- Group (Attributes attributes)

  Constructs a Group object.
- ~Group ()

  Destructs a Group object.
- std::string getClass () const override

  Gets the type of the shape.
- Attributes getAttributes () const

  Gets the attributes of the shape.
- void addElement (SVGElement *shape) override

  Adds a shape to the composite group.
- std::vector< SVGElement * > getElements () const

  Gets the vector of shapes in the composite group.
- void printData () const override

  Prints the data of the shape.

Private Attributes

- std::vector< SVGElement * > shapes

  Vector of shapes in the group.
- Attributes attributes

  Attributes of the group.

Additional Inherited Members

3.5.1 Detailed Description

A composite class that contains a vector of shape pointers (polymorphic).

The Group class is derived from the SVGElement class and defines a group of SVGElements. The Group class is a composite class that contains a vector of SVGElement pointers (polymorphic). The Group class is used to group SVGElements together.

Definition at line 20 of file Group.hpp.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Group()

```
Group::Group (
            Attributes attributes )
```

Constructs a Group object.

Parameters

attributes	The attributes of the group.

Definition at line 5 of file Group.cpp.

```
5 : attributes(attributes) {}
```

3.5.3 Member Function Documentation

3.5.3.1 addElement()

```
void Group::addElement (
            SVGElement * shape )  [override], [virtual]
```

Adds a shape to the composite group.

Parameters

shape	The shape to be added to the composite group.

Reimplemented from SVGElement.

Definition at line 17 of file Group.cpp.

```
17                                      {
18     shapes.push_back(shape);
19     shape->setParent(this);
20 }
```

3.5.3.2 getAttributes()

```
Attributes Group::getAttributes ( ) const
```

Gets the attributes of the shape.

Note

This function uses rapidXML to parse the SVG file and get the attributes of the shape.

Returns

The attributes of the shape that parsed from the SVG file.

Definition at line 15 of file Group.cpp.

```
15 { return attributes; }
```

3.5.3.3 getClass()

```
std::string Group::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string that represents the type of the shape.

Implements SVGElement.

Definition at line 13 of file Group.cpp.

```
13 { return "Group"; }
```

3.5.3.4 getElements()

```
std::vector< SVGElement * > Group::getElements ( ) const
```

Gets the vector of shapes in the composite group.

Returns

The vector of shapes in the composite group.

Definition at line 22 of file Group.cpp.

```
22 { return shapes; }
```

3.5.3.5 printData()

```
void Group::printData ( ) const  [override], [virtual]
```

Prints the data of the shape.

Note

This function is used for debugging purposes.

Reimplemented from SVGElement.

Definition at line 24 of file Group.cpp.

```
24                              {
25     std::cout « "Group: " « std::endl;
26     for (auto shape : shapes) {
27         std::cout « "    ";
28         shape->printData();
29         std::cout « std::endl;
30     }
31 }
```

The documentation for this class was generated from the following files:

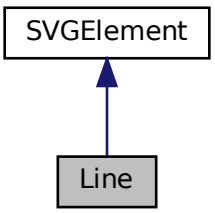
- src/graphics/Group.hpp
- src/graphics/Group.cpp

3.6 Line Class Reference

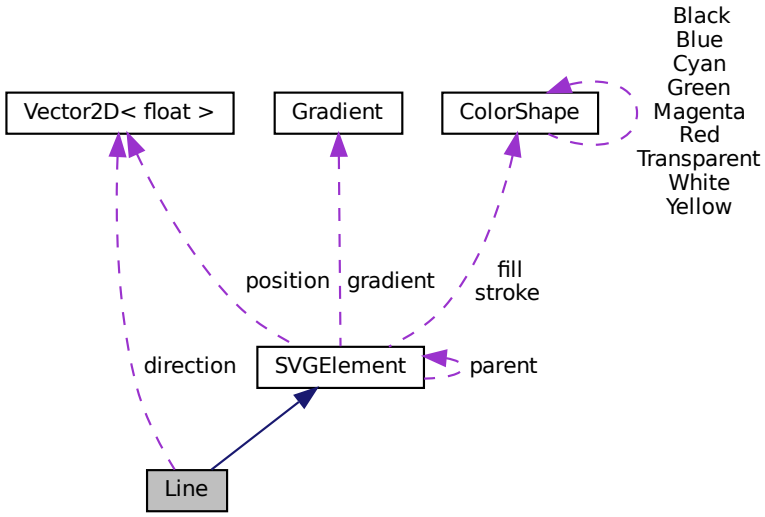
Represents a line in 2D space.

```
#include <Line.hpp>
```

Inheritance diagram for Line:



Collaboration diagram for Line:



Public Member Functions

- Line (const Vector2Df &point1, const Vector2Df &point2, ColorShape stroke, float stroke_width)

  Constructs a Line object.
- std::string getClass () const override

  Gets the type of the shape.
- void setDirection (const Vector2Df &direction)

  Sets the direction of the line.
- Vector2Df getDirection () const

  Gets the direction of the line.
- float getLength () const

  Gets the length of the line.

Private Attributes

- Vector2Df direction

  Direction of the line.

Additional Inherited Members

3.6.1 Detailed Description

Represents a line in 2D space.

The Line class is derived from the SVGElement class and defines a line segment with a specified direction and thickness.

Definition at line 12 of file Line.hpp.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Line()

```
Line::Line (
            const Vector2Df & point1,
            const Vector2Df & point2,
            ColorShape stroke,
            float stroke_width )
```

Constructs a Line object.

Parameters

point1	The starting point of the line.
point2	The ending point of the line.
stroke	The color of the line (default is sf::Color::White).
stroke_width	The thickness of the line (default is 1.0).

Definition at line 5 of file Line.cpp.

```
7     : SVGElement(ColorShape::Transparent, stroke, stroke_width, point1),
8       direction(point2) {}
```

3.6.3 Member Function Documentation

3.6.3.1 getClass()

```
std::string Line::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Line".

Implements SVGElement.

Definition at line 10 of file Line.cpp.

```
10 { return "Line"; }
```

3.6.3.2 getDirection()

```
Vector2Df Line::getDirection ( ) const
```

Gets the direction of the line.

Returns

The direction of the line.

Definition at line 16 of file Line.cpp.

```
16 { return direction; }
```

3.6.3.3 getLength()

```
float Line::getLength ( ) const
```

Gets the length of the line.

Returns

The length of the line.

Definition at line 18 of file Line.cpp.

```
18                              {
19     return std::sqrt(direction.x * direction.x + direction.y * direction.y);
20 }
```

3.6.3.4 setDirection()

```
void Line::setDirection (
            const Vector2Df & direction )
```

Sets the direction of the line.

Parameters

direction	The new direction of the line.

Definition at line 12 of file Line.cpp.

```
12                                              {
13     this->direction = direction;
14 }
```

The documentation for this class was generated from the following files:

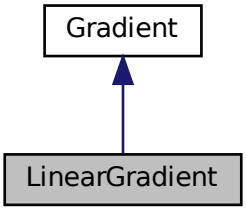
- src/graphics/Line.hpp
- src/graphics/Line.cpp

3.7 LinearGradient Class Reference

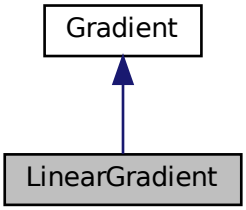
A class that represents a linear gradient.

`#include <LinearGradient.hpp>`

Inheritance diagram for LinearGradient:



Collaboration diagram for LinearGradient:



Public Member Functions

- LinearGradient (std::vector< Stop > stops, std::pair< Vector2Df, Vector2Df > points, std::string units)
  Constructs a LinearGradient object.
- std::string getClass () const override
  Gets the type of the gradient.

3.7.1 Detailed Description

A class that represents a linear gradient.

The LinearGradient class is derived from the Gradient class and represents a linear gradient. It contains a vector of Stop objects that represent the the stops of the gradient. It also contains a pair of Vector2D objects that represent the start and end points of the gradient.

Definition at line 14 of file LinearGradient.hpp.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 LinearGradient()

```
LinearGradient::LinearGradient (
            std::vector< Stop > stops,
            std::pair< Vector2Df, Vector2Df > points,
            std::string units )
```

Constructs a LinearGradient object.

Parameters

stops	The stops of the gradient.
points	The start and end points of the gradient.
units	The units of the gradient.

Definition at line 3 of file LinearGradient.cpp.

```
6     : Gradient(stops, points, units) {}
```

3.7.3 Member Function Documentation

3.7.3.1 getClass()

```
std::string LinearGradient::getClass ( ) const  [override], [virtual]
```

Gets the type of the gradient.

Returns

The string "LinearGradient".

Note

This function is used for determining the type of the gradient.

Implements Gradient.

Definition at line 8 of file LinearGradient.cpp.

```
8 { return "LinearGradient"; }
```

The documentation for this class was generated from the following files:

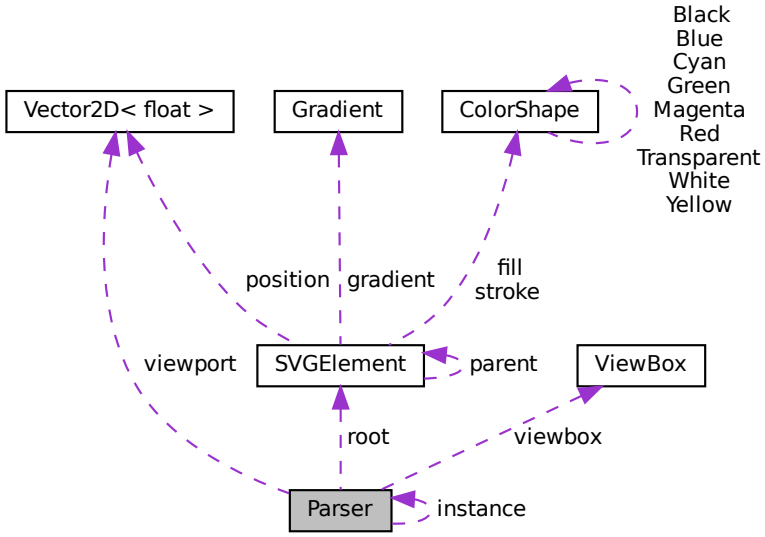
- src/graphics/LinearGradient.hpp
- src/graphics/LinearGradient.cpp

3.8 Parser Class Reference

To manipulate and parse an SVG file.

`#include <Parser.hpp>`

Collaboration diagram for Parser:



Public Member Functions

- Parser (const Parser &)=delete

  Deleted copy constructor to enforce the singleton pattern.
- ~Parser ()

  Destructor for the Parser class.
- Group * getRoot ()

  Gets the root of the tree of SVGElements.
- void printShapesData ()

  Prints the data of the shapes.
- ViewBox getViewBox () const

  Gets the viewbox of the SVG file.
- Vector2Df getViewPort () const

  Gets the viewport of the SVG file.

Static Public Member Functions

- static Parser * getInstance (const std::string &file_name)

  Gets the singleton instance of the Parser class.

Private Member Functions

- Parser (const std::string &file_name)

  Construct a new Parser object.
- SVGElement * parseElements (std::string file_name)

  Parses the SVG file and creates a tree of SVGElements.
- std::string getAttribute (rapidxml::xml_node<> *node, std::string name)

  Gets the attributes of a node.
- float getFloatAttribute (rapidxml::xml_node<> *node, std::string name)

  Gets the floating point attributes of a node.
- std::vector< Stop > getGradientStops (rapidxml::xml_node<> *node)

  Gets the gradient stops of a node.
- void GetGradients (rapidxml::xml_node<> *node)

  Gets the gradients of a node.
- Gradient * parseGradient (std::string id)

  Gets the gradient of a node.
- ColorShape parseColor (rapidxml::xml_node<> *node, std::string color, std::string &id)

  Gets the color attributes of a node.
- std::vector< Vector2Df > parsePoints (rapidxml::xml_node<> *node)

  Gets the points of the element.
- std::vector< PathPoint > parsePathPoints (rapidxml::xml_node<> *node)

  Gets the points of the path element.
- std::vector< std::string > getTransformOrder (rapidxml::xml_node<> *node)

  Gets the transform order of the element.
- Line * parseLine (rapidxml::xml_node<> *node, const ColorShape &stroke_color, float stroke_width)

  Parses the line element.
- Rect * parseRect (rapidxml::xml_node<> *node, const ColorShape &fill_color, const ColorShape &stroke↩_color, float stroke_width)

  Parses the rect element.
- class Plyline * parsePolyline (rapidxml::xml_node<> *node, const ColorShape &fill_color, const ColorShape &stroke_color, float stroke_width)

  Parses the polyline element.
- class Plygon * parsePolygon (rapidxml::xml_node<> *node, const ColorShape &fill_color, const ColorShape &stroke_color, float stroke_width)

  Parses the polygon element.
- Circle * parseCircle (rapidxml::xml_node<> *node, const ColorShape &fill_color, const ColorShape &stroke_color, float stroke_width)

  Parses the circle element.
- class Ell * parseEllipse (rapidxml::xml_node<> *node, const ColorShape &fill_color, const ColorShape &stroke_color, float stroke_width)

  Parses the ellipse element.
- Path * parsePath (rapidxml::xml_node<> *node, const ColorShape &fill_color, const ColorShape &stroke↩_color, float stroke_width)

  Parses the path element.
- Text * parseText (rapidxml::xml_node<> *node, const ColorShape &fill_color, const ColorShape &stroke↩_color, float stroke_width)

  Parses the text element.
- SVGElement * parseShape (rapidxml::xml_node<> *node)

  Parses the group of elements.

Private Attributes

- SVGElement * root

  The root of the SVG file.
- std::map< std::string, Gradient * > gradients
- ViewBox viewbox

  The viewbox of the SVG file.
- Vector2Df viewport

  The viewport of the SVG file.

Static Private Attributes

- static Parser * instance = nullptr

  The instance of the Parser.

3.8.1 Detailed Description

To manipulate and parse an SVG file.

The Parser class is a singleton class that is used to parse an SVG file and create a tree of SVGElements.

Definition at line 24 of file Parser.hpp.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Parser()

```
Parser::Parser (
            const std::string & file_name )  [private]
```

Construct a new Parser object.

Parameters

file_name	The name of the file to be parsed.

Definition at line 179 of file Parser.cpp.

```
179                                         {
180     root = parseElements(file_name);
181 }
```

3.8.3 Member Function Documentation

3.8.3.1 getAttribute()

```
std::string Parser::getAttribute (
            rapidxml::xml_node<> * node,
            std::string name )  [private]
```

Gets the attributes of a node.

Parameters

node	The node to be parsed.
name	The name of tag to be parsed.

Returns

The attributes of the node.

Definition at line 328 of file Parser.cpp.

```
328                                                                         {
329     if (name == "text") return removeExtraSpaces(node->value());
330     std::string result;
331     if (node->first_attribute(name.c_str()) == NULL) {
332         if (name == "fill" || name == "stop-color")
333             result = "black";
334         else if (name == "stroke" || name == "transform" || name == "rotate" ||
335                  name == "font-style")
336             result = "none";
337         else if (name == "text-anchor")
338             result = "start";
339         else if (name == "fill-rule")
340             result = "nonzero";
341         else if (name == "gradientUnits")
342             result = "objectBoundingBox";
343     } else {
344         result = node->first_attribute(name.c_str())->value();
345     }
346     return result;
347 }
```

3.8.3.2 getFloatAttribute()

```
float Parser::getFloatAttribute (
            rapidxml::xml_node<> * node,
            std::string name )  [private]
```

Gets the floating point attributes of a node.

Parameters

node	The node to be parsed.
name	The name of tag to be parsed.

Returns

The floating point attributes of the node.

Definition at line 351 of file Parser.cpp.

```
351                                                                         {
352     float result;
353     if (node->first_attribute(name.c_str()) == NULL) {
354         if (std::string(node->name()).find("Gradient") != std::string::npos) {
355             // Handle gradient-specific attribute default values
356             if (name == "x1" || name == "y1" || name == "fr")
357                 result = 0;
358             else if (name == "cx" || name == "cy")
359                 result = name == "cx" ? 0.5 * this->viewbox.getWidth()
360                                       : 0.5 * this->viewbox.getHeight();
361             else if (name == "r") {
362                 result = sqrt((pow(this->viewbox.getWidth(), 2) +
363                                pow(this->viewbox.getHeight(), 2)) /
364                               2) /
365                          2;
366             } else if (name == "fx" || name == "fy")
367                 result = name == "fx" ? getFloatAttribute(node, "cx")
368                                       : getFloatAttribute(node, "cy");
369             else
370                 result = name == "x2" ? this->viewbox.getWidth()
371                                       : this->viewbox.getHeight();
372         } else {
373             // Handle default float attribute values for other elements
374             if (name == "stroke-width" || name == "stroke-opacity" ||
375                 name == "fill-opacity" || name == "opacity" ||
376                 name == "stop-opacity")
377                 result = 1;
378             else
379                 result = 0;
380         }
381     } else {
382         if (name == "width" || name == "height") {
383             // Handle width and height attributes with percentage or point units
384             std::string value = node->first_attribute(name.c_str())->value();
385             if (value.find("%") != std::string::npos) {
386                 result = std::stof(value.substr(0, value.find("%"))) *
387                          this->viewbox.getWidth() / 100;
388             } else if (value.find("pt") != std::string::npos) {
389                 result = std::stof(value.substr(0, value.find("pt"))) * 1.33;
390             } else {
391                 result = std::stof(value);
392             }
393         } else
394             result = std::stof(node->first_attribute(name.c_str())->value());
395     }
396     return result;
397 }
```

3.8.3.3 GetGradients()

```
void Parser::GetGradients (
            rapidxml::xml_node<> * node )  [private]
```

Gets the gradients of a node.

Parameters

node	The node to be parsed.

Definition at line 473 of file Parser.cpp.

```
473                                                 {
474     rapidxml::xml_node<> *gradient_node = node->first_node();
475     while (gradient_node) {
476         if (std::string(gradient_node->name()).find("Gradient") !=
477             std::string::npos) {
478             Gradient *gradient;
479             std::string id = getAttribute(gradient_node, "id");
480             std::string units = getAttribute(gradient_node, "gradientUnits");
481             std::vector< Stop > stops = getGradientStops(gradient_node);
482             std::string href = getAttribute(gradient_node, "xlink:href");
483             int pos = href.find("#");
484             if (pos != std::string::npos) {
485                 href = href.substr(pos + 1);
```

```
486             }
487             if (std::string(gradient_node->name()).find("linear") !=
488                 std::string::npos) {
489                 float x1 = getFloatAttribute(gradient_node, "x1");
490                 float y1 = getFloatAttribute(gradient_node, "y1");
491                 float x2 = getFloatAttribute(gradient_node, "x2");
492                 float y2 = getFloatAttribute(gradient_node, "y2");
493                 std::pair< Vector2Df, Vector2Df > points = {{x1, y1}, {x2, y2}};
494                 gradient = new LinearGradient(stops, points, units);
495                 if (this->gradients.find(id) == this->gradients.end())
496                     this->gradients[id] = gradient;
497             } else if (std::string(gradient_node->name()).find("radial") !=
498                        std::string::npos) {
499                 float cx = getFloatAttribute(gradient_node, "cx");
500                 float cy = getFloatAttribute(gradient_node, "cy");
501                 float fx = getFloatAttribute(gradient_node, "fx");
502                 float fy = getFloatAttribute(gradient_node, "fy");
503                 float r = getFloatAttribute(gradient_node, "r");
504                 float fr = getFloatAttribute(gradient_node, "fr");
505                 std::pair< Vector2Df, Vector2Df > points = {{cx, cy}, {fx, fy}};
506                 Vector2Df radius(r, fr);
507                 gradient = new RadialGradient(stops, points, radius, units);
508                 if (this->gradients.find(id) == this->gradients.end())
509                     this->gradients[id] = gradient;
510             }
511             if (href != "") {
512                 for (auto stop : parseGradient(href)->getStops()) {
513                     gradient->addStop(stop);
514                 }
515             }
516             if (gradient != NULL)
517                 gradient->setTransforms(getTransformOrder(gradient_node));
518         }
519         gradient_node = gradient_node->next_sibling();
520     }
521 }
```

3.8.3.4 getGradientStops()

```
std::vector< Stop > Parser::getGradientStops (
            rapidxml::xml_node<> * node )  [private]
```

Gets the gradient stops of a node.

Parameters

node	The node to be parsed.

Returns

The gradient stops of the node.

Definition at line 456 of file Parser.cpp.

```
456                                                                     {
457     std::vector< Stop > stops;
458     rapidxml::xml_node<> *stop_node = node->first_node();
459     while (stop_node) {
460         if (std::string(stop_node->name()) == "stop") {
461             std::string id = "";
462             ColorShape color = parseColor(stop_node, "stop-color", id);
463             float offset = getFloatAttribute(stop_node, "offset");
464             if (offset > 1) offset /= 100;
465             stops.push_back(Stop(color, offset));
466         }
467         stop_node = stop_node->next_sibling();
468     }
469     return stops;
470 }
```

3.8.3.5 getInstance()

```
Parser * Parser::getInstance (
            const std::string & file_name )  [static]
```

Gets the singleton instance of the Parser class.

Parameters

file_name	The name of the file to be parsed.

Returns

The singleton instance of the Parser class.

Definition at line 171 of file Parser.cpp.

```
171                                                     {
172     if (instance == nullptr) {
173         instance = new Parser(file_name);
174     }
175     return instance;
176 }
```

3.8.3.6 getRoot()

```
Group * Parser::getRoot ( )
```

Gets the root of the tree of SVGElements.

Returns

The root of the tree of SVGElements.

Definition at line 184 of file Parser.cpp.

```
184 { return dynamic_cast< Group * >(root); }
```

3.8.3.7 getTransformOrder()

```
std::vector< std::string > Parser::getTransformOrder (
            rapidxml::xml_node<> * node )  [private]
```

Gets the transform order of the element.

Parameters

node	The node to be parsed.

Returns

The transform order of the element

Definition at line 694 of file Parser.cpp.

```
695                                 {
696     std::string transform_tag;
697     if (std::string(node->name()).find("Gradient") != std::string::npos) {
698         transform_tag = getAttribute(node, "gradientTransform");
699     } else {
700         transform_tag = getAttribute(node, "transform");
701     }
702 
703     std::vector< std::string > order;
704     std::stringstream ss(transform_tag);
705     std::string type;
706     while (ss » type) {
707         if (type.find("translate") != std::string::npos ||
708             type.find("scale") != std::string::npos ||
709             type.find("rotate") != std::string::npos ||
710             type.find("matrix") != std::string::npos) {
711             while (type.find(")") == std::string::npos) {
712                 std::string temp;
713                 ss » temp;
714                 type += " " + temp;
715             }
716             std::string temp = type.substr(0, type.find("(") + 1);
717             temp.erase(std::remove(temp.begin(), temp.end(), ' '), temp.end());
718             type.erase(0, type.find("(") + 1);
719             type = temp + type;
720             order.push_back(type);
721         }
722     }
723     return order;
724 }
```

3.8.3.8 getViewBox()

```
ViewBox Parser::getViewBox ( ) const
```

Gets the viewbox of the SVG file.

Returns

The viewbox of the SVG file.

Definition at line 911 of file Parser.cpp.

```
911 { return viewbox; }
```

3.8.3.9 getViewPort()

```
Vector2Df Parser::getViewPort ( ) const
```

Gets the viewport of the SVG file.

Returns

The viewport of the SVG file.

Definition at line 914 of file Parser.cpp.

```
914 { return viewport; }
```

3.8.3.10 parseCircle()

```
Circle * Parser::parseCircle (
            rapidxml::xml_node<> * node,
            const ColorShape & fill_color,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the circle element.

Parameters

node	The node to be parsed.
fill_color	The color of the fill
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The circle element

Definition at line 799 of file Parser.cpp.

```
802                                                 {
803     float cx = getFloatAttribute(node, "cx");
804     float cy = getFloatAttribute(node, "cy");
805     float radius = getFloatAttribute(node, "r");
806     Circle *shape = new Circle(radius, Vector2Df(cx, cy), fill_color,
807                                stroke_color, stroke_width);
808     return shape;
809 }
```

3.8.3.11 parseColor()

```
ColorShape Parser::parseColor (
            rapidxml::xml_node<> * node,
            std::string color,
            std::string & id )  [private]
```

Gets the color attributes of a node.

Parameters

node	The node to be parsed.
color	The name of the color tag to be parsed.
id	The id to check if the color is a reference.

Returns

The color attributes of the node.

Definition at line 400 of file Parser.cpp.

```
401                                         {
```

```
402     std::string color = getAttribute(node, name);
403     color.erase(std::remove(color.begin(), color.end(), ' '), color.end());
404     if (color.find("url") == std::string::npos) {
405         for (auto &c : color) c = tolower(c);
406     }
407     if (color == "none")
408         return ColorShape::Transparent;
409     else {
410         ColorShape result;
411         if (color.find("url") != std::string::npos) {
412             // Handle gradient color reference
413             if (color.find("'") != std::string::npos) {
414                 id = color.substr(color.find("'") + 1);
415                 id.erase(id.find("'"));
416                 id.erase(id.find("#"), 1);
417             } else {
418                 id = color.substr(color.find("#") + 1);
419                 id.erase(id.find(")"));
420             }
421             result = ColorShape::Transparent;
422         } else if (color.find("#") != std::string::npos) {
423             // Handle hex color representation
424             result = getHexColor(color);
425         } else if (color.find("rgb") != std::string::npos) {
426             // Handle RGB color representation
427             result = getRgbColor(color);
428         } else {
429             // Handle predefined color names
430             auto color_code = color_map.find(color);
431             if (color_code == color_map.end()) {
432                 std::cout « "Color " « color « " not found" « std::endl;
433                 exit(-1);
434             }
435             result = color_code->second;
436         }
437         if (name == "stop-color")
438             result.a = result.a * getFloatAttribute(node, "stop-opacity");
439         else
440             result.a = result.a * getFloatAttribute(node, name + "-opacity") *
441                        getFloatAttribute(node, "opacity");
442         return result;
443     }
444 }
```

3.8.3.12 parseElements()

```
SVGElement * Parser::parseElements (
            std::string file_name )  [private]
```

Parses the SVG file and creates a tree of SVGElements.

Parameters

file_name	The name of the file to be parsed.

Returns

The root of the tree of SVGElements.

Definition at line 198 of file Parser.cpp.

```
198                                                 {
199     rapidxml::xml_document<> doc;
200     std::ifstream file(file_name);
201     std::vector< char > buffer((std::istreambuf_iterator< char >(file)),
202                                std::istreambuf_iterator< char >());
203     buffer.push_back('\0');
204     doc.parse< 0 >(&buffer[0]);
205
206     rapidxml::xml_node<> *svg = doc.first_node();
207     viewport.x = getFloatAttribute(svg, "width");
```

```
208     viewport.y = getFloatAttribute(svg, "height");
209     std::string viewbox = getAttribute(svg, "viewBox");
210     if (viewbox != "") {
211         std::stringstream ss(viewbox);
212         float x, y, w, h;
213         ss » x » y » w » h;
214         this->viewbox = ViewBox(x, y, w, h);
215     }
216     rapidxml::xml_node<> *node = svg->first_node();
217     rapidxml::xml_node<> *prev = NULL;
218 
219     SVGElement *root = new Group();
220     SVGElement *current = root;
221 
222     // Parse SVG elements
223     while (node) {
224         if (std::string(node->name()) == "defs") {
225             // Parse gradients
226             GetGradients(node);
227             prev = node;
228             node = node->next_sibling();
229         } else if (std::string(node->name()) == "g") {
230             // Parse Group attributes
231             Group *group = dynamic_cast< Group * >(current);
232             for (auto group_attribute : group->getAttributes()) {
233                 bool found = false;
234                 for (auto attribute = node->first_attribute(); attribute;
235                      attribute = attribute->next_attribute()) {
236                     if (std::string(attribute->name()) ==
237                         group_attribute.first) {
238                         if (group_attribute.first == "opacity") {
239                             // Adjust opacity if already present in the group
240                             // and node
241                             std::string opacity = std::to_string(
242                                 std::stof(attribute->value()) *
243                                 std::stof(group_attribute.second));
244                             char *value = doc.allocate_string(opacity.c_str());
245                             attribute->value(value);
246                         }
247                         found = true;
248                         break;
249                     }
250                 }
251 
252                 if (!found && group_attribute.first != "transform") {
253                     // Add missing attributes from the group to the node
254                     char *name =
255                         doc.allocate_string(group_attribute.first.c_str());
256                     char *value =
257                         doc.allocate_string(group_attribute.second.c_str());
258                     rapidxml::xml_attribute<> *new_attribute =
259                         doc.allocate_attribute(name, value);
260                     node->append_attribute(new_attribute);
261                 }
262             }
263 
264             Group *new_group = new Group(xmlToString(node->first_attribute()));
265             new_group->setTransforms(getTransformOrder(node));
266             current->addElement(new_group);
267             current = new_group;
268             prev = node;
269             node = node->first_node();
270         } else {
271             // Parse Shape attributes and add to current group
272             Group *group = dynamic_cast< Group * >(current);
273 
274             for (auto group_attribute : group->getAttributes()) {
275                 bool found = false;
276                 for (auto attribute = node->first_attribute(); attribute;
277                      attribute = attribute->next_attribute()) {
278                     if (std::string(attribute->name()) ==
279                         group_attribute.first) {
280                         if (group_attribute.first == "opacity") {
281                             std::string opacity = std::to_string(
282                                 std::stof(attribute->value()) *
283                                 std::stof(group_attribute.second));
284                             char *value = doc.allocate_string(opacity.c_str());
285                             attribute->value(value);
286                         }
287                         found = true;
288                         break;
289                     }
290                 }
291 
292                 if (!found && group_attribute.first != "transform") {
293                     char *name =
294                         doc.allocate_string(group_attribute.first.c_str());
```

```
295                     char *value =
296                         doc.allocate_string(group_attribute.second.c_str());
297                     rapidxml::xml_attribute<> *new_attribute =
298                         doc.allocate_attribute(name, value);
299                     node->append_attribute(new_attribute);
300                 }
301             }
302
303             SVGElement *shape = parseShape(node);
304             if (shape != NULL) current->addElement(shape);
305             prev = node;
306             node = node->next_sibling();
307         }
308
309         if (node == NULL && current != root) {
310             while (prev->parent()->next_sibling() == NULL) {
311                 current = current->getParent();
312                 prev = prev->parent();
313                 if (prev == svg) {
314                     break;
315                 }
316             }
317             if (prev == svg) {
318                 break;
319             }
320             current = current->getParent();
321             node = prev->parent()->next_sibling();
322         }
323     }
324     return root;
325 }
```

3.8.3.13 parseEllipse()

```
Ell * Parser::parseEllipse (
            rapidxml::xml_node<> * node,
            const ColorShape & fill_color,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the ellipse element.

Parameters

node	The node to be parsed.
fill_color	The color of the fill
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The ellipse element

Definition at line 812 of file Parser.cpp.

```
814                                                                         {
815     float radius_x = getFloatAttribute(node, "rx");
816     float radius_y = getFloatAttribute(node, "ry");
817     float cx = getFloatAttribute(node, "cx");
818     float cy = getFloatAttribute(node, "cy");
819     Ell *shape = new Ell(Vector2Df(radius_x, radius_y), Vector2Df(cx, cy),
820                          fill_color, stroke_color, stroke_width);
821     return shape;
822 }
```

3.8.3.14 parseGradient()

```
Gradient * Parser::parseGradient (
            std::string id )  [private]
```

Gets the gradient of a node.

Parameters

id	The id of the gradient to be parsed.

Returns

The gradient of the node.

Definition at line 447 of file Parser.cpp.

```
447                                                 {
448     if (gradients.find(id) == gradients.end()) {
449         std::cout « "Gradient " « id « " not found" « std::endl;
450         exit(-1);
451     }
452     return gradients.at(id);
453 }
```

3.8.3.15 parseLine()

```
Line * Parser::parseLine (
            rapidxml::xml_node<> * node,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the line element.

Parameters

node	The node to be parsed.
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The line element

Definition at line 774 of file Parser.cpp.

```
775                                                                                 {
776     Line *shape = new Line(
777         Vector2Df(getFloatAttribute(node, "x1"), getFloatAttribute(node, "y1")),
778         Vector2Df(getFloatAttribute(node, "x2"), getFloatAttribute(node, "y2")),
779         stroke_color, stroke_width);
780     return shape;
781 }
```

3.8.3.16 parsePath()

```
Path * Parser::parsePath (
            rapidxml::xml_node<> * node,
            const ColorShape & fill_color,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the path element.

Parameters

node	The node to be parsed.
fill_color	The color of the fill
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The path element

Definition at line 884 of file Parser.cpp.

```
886                                                                                 {
887     Path *shape = new Path(fill_color, stroke_color, stroke_width);
888     std::vector< PathPoint > points = parsePathPoints(node);
889     for (auto point : points) {
890         shape->addPoint(point);
891     }
892     std::string fill_rule = getAttribute(node, "fill-rule");
893     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
894                     fill_rule.end());
895     shape->setFillRule(fill_rule);
896     return shape;
897 }
```

3.8.3.17 parsePathPoints()

```
std::vector< PathPoint > Parser::parsePathPoints (
            rapidxml::xml_node<> * node )  [private]
```

Gets the points of the path element.

Parameters

node	The node to be parsed.

Returns

The points of the path element

Definition at line 541 of file Parser.cpp.

```
541                                                                     {
542     std::vector< PathPoint > points;
543     std::string path_string = getAttribute(node, "d");
```

```
544
545     // Pre-processing the raw path string
546     formatSvgPathString(path_string);
547
548     // Tokenizing the path string using stringstream
549     std::stringstream ss(path_string);
550     std::string element;
551     PathPoint pPoint{{0, 0}, 'M'};  // Default starting point and command
552     while (ss » element) {
553         if (std::isalpha(element[0])) {
554             pPoint.tc = element[0];
555             if (tolower(pPoint.tc) == 'm' || tolower(pPoint.tc) == 'l' ||
556                 tolower(pPoint.tc) == 'c' || tolower(pPoint.tc) == 's' ||
557                 tolower(pPoint.tc) == 'q' || tolower(pPoint.tc) == 't')
558                 ss » pPoint.point.x » pPoint.point.y;
559             else if (tolower(pPoint.tc) == 'h') {
560                 ss » pPoint.point.x;
561                 pPoint.point.y = 0;
562             } else if (tolower(pPoint.tc) == 'v') {
563                 ss » pPoint.point.y;
564                 pPoint.point.x = 0;
565             } else if (tolower(pPoint.tc) == 'a') {
566                 ss » pPoint.radius.x » pPoint.radius.y;
567                 ss » pPoint.x_axis_rotation;
568                 ss » pPoint.large_arc_flag » pPoint.sweep_flag;
569                 ss » pPoint.point.x » pPoint.point.y;
570             }
571         } else {
572             if (tolower(pPoint.tc) == 'm' || tolower(pPoint.tc) == 'l' ||
573                 tolower(pPoint.tc) == 'c' || tolower(pPoint.tc) == 's' ||
574                 tolower(pPoint.tc) == 'q' || tolower(pPoint.tc) == 't') {
575                 if (tolower(pPoint.tc) == 'm') pPoint.tc = 'L';
576                 pPoint.point.x = std::stof(element);
577                 ss » pPoint.point.y;
578             } else if (tolower(pPoint.tc) == 'h') {
579                 pPoint.point.x = std::stof(element);
580                 pPoint.point.y = 0;
581             } else if (tolower(pPoint.tc) == 'v') {
582                 pPoint.point.y = std::stof(element);
583                 pPoint.point.x = 0;
584             } else if (tolower(pPoint.tc) == 'a') {
585                 pPoint.radius.x = std::stof(element);
586                 ss » pPoint.radius.y;
587                 ss » pPoint.x_axis_rotation;
588                 ss » pPoint.large_arc_flag » pPoint.sweep_flag;
589                 ss » pPoint.point.x » pPoint.point.y;
590             }
591         }
592         points.push_back(pPoint);
593     }
594
595     std::vector< PathPoint > handle_points;
596
597     // Processing and transforming raw path points
598     Vector2Df first_point{0, 0}, cur_point{0, 0};
599     int n = points.size();
600     for (int i = 0; i < n; i++) {
601         if (tolower(points[i].tc) == 'm') {
602             first_point = points[i].point;
603             if (points[i].tc == 'm') {
604                 first_point.x = cur_point.x + points[i].point.x;
605                 first_point.y = cur_point.y + points[i].point.y;
606             }
607             cur_point = first_point;
608             handle_points.push_back({first_point, 'm'});
609
610         } else if (tolower(points[i].tc) == 'l' ||
611                    tolower(points[i].tc) == 't') {
612             Vector2Df end_point{cur_point.x + points[i].point.x,
613                                 cur_point.y + points[i].point.y};
614             if (points[i].tc == 'L' || points[i].tc == 'T')
615                 end_point = points[i].point;
616             cur_point = end_point;
617             char TC = tolower(points[i].tc);
618             handle_points.push_back({end_point, TC});
619
620         } else if (tolower(points[i].tc) == 'h') {
621             Vector2Df end_point{cur_point.x + points[i].point.x, cur_point.y};
622             if (points[i].tc == 'H')
623                 end_point = Vector2Df{points[i].point.x, cur_point.y};
624             cur_point = end_point;
625             handle_points.push_back({end_point, 'h'});
626
627         } else if (tolower(points[i].tc) == 'v') {
628             Vector2Df end_point{cur_point.x, cur_point.y + points[i].point.y};
629             if (points[i].tc == 'V')
630                 end_point = Vector2Df{cur_point.x, points[i].point.y};
```

```
631             cur_point = end_point;
632             handle_points.push_back({end_point, 'v'});
633 
634         } else if (tolower(points[i].tc) == 'c') {
635             if (i + 2 < n) {
636                 Vector2Df control_point1 =
637                     Vector2Df{cur_point.x + points[i].point.x,
638                               cur_point.y + points[i].point.y};
639                 Vector2Df control_point2 =
640                     Vector2Df{cur_point.x + points[i + 1].point.x,
641                               cur_point.y + points[i + 1].point.y};
642                 Vector2Df control_point3 =
643                     Vector2Df{cur_point.x + points[i + 2].point.x,
644                               cur_point.y + points[i + 2].point.y};
645                 if (points[i].tc == 'C') {
646                     control_point1 = points[i].point;
647                     control_point2 = points[i + 1].point;
648                     control_point3 = points[i + 2].point;
649                 }
650                 i += 2;
651                 cur_point = control_point3;
652                 handle_points.push_back({control_point1, 'c'});
653                 handle_points.push_back({control_point2, 'c'});
654                 handle_points.push_back({control_point3, 'c'});
655             }
656         } else if (tolower(points[i].tc) == 'z') {
657             cur_point = first_point;
658             handle_points.push_back({first_point, 'z'});
659 
660         } else if (tolower(points[i].tc) == 's' ||
661                    tolower(points[i].tc) == 'q') {
662             if (i + 1 < n) {
663                 Vector2Df control_point1 =
664                     Vector2Df{cur_point.x + points[i].point.x,
665                               cur_point.y + points[i].point.y};
666                 Vector2Df control_point2 =
667                     Vector2Df{cur_point.x + points[i + 1].point.x,
668                               cur_point.y + points[i + 1].point.y};
669                 if (points[i].tc == 'S' || points[i].tc == 'Q') {
670                     control_point1 = points[i].point;
671                     control_point2 = points[i + 1].point;
672                 }
673                 i += 1;
674                 cur_point = control_point2;
675                 char TC = tolower(points[i].tc);
676                 handle_points.push_back({control_point1, TC});
677                 handle_points.push_back({control_point2, TC});
678             }
679 
680         } else if (tolower(points[i].tc) == 'a') {
681             Vector2Df end_point{cur_point.x + points[i].point.x,
682                                 cur_point.y + points[i].point.y};
683             if (points[i].tc == 'A') end_point = points[i].point;
684             handle_points.push_back(
685                 {end_point, 'a', points[i].radius, points[i].x_axis_rotation,
686                  points[i].large_arc_flag, points[i].sweep_flag});
687             cur_point = end_point;
688         }
689     }
690     return handle_points;
691 }
```

3.8.3.18 parsePoints()

```
std::vector< Vector2Df > Parser::parsePoints (
            rapidxml::xml_node<> * node )  [private]
```

Gets the points of the element.

Parameters

node	The node to be parsed.

Returns

The points of the element

Definition at line 524 of file Parser.cpp.

```
524                                                                         {
525     std::vector< Vector2Df > points;
526     std::string points_string = getAttribute(node, "points");
527 
528     std::stringstream ss(points_string);
529     float x, y;
530 
531     while (ss » x) {
532         if (ss.peek() == ',') ss.ignore();
533         ss » y;
534         points.push_back(Vector2Df(x, y));
535     }
536 
537     return points;
538 }
```

3.8.3.19 parsePolygon()

```
Plygon * Parser::parsePolygon (
            rapidxml::xml_node<> * node,
            const ColorShape & fill_color,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the polygon element.

Parameters

node	The node to be parsed.
fill_color	The color of the fill
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The polygon element

Definition at line 825 of file Parser.cpp.

```
828                                                 {
829     Plygon *shape = new Plygon(fill_color, stroke_color, stroke_width);
830     std::vector< Vector2Df > points = parsePoints(node);
831     for (auto point : points) {
832         shape->addPoint(point);
833     }
834     std::string fill_rule = getAttribute(node, "fill-rule");
835     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
836                     fill_rule.end());
837     shape->setFillRule(fill_rule);
838     return shape;
839 }
```

3.8.3.20 parsePolyline()

```
Pyline * Parser::parsePolyline (
            rapidxml::xml_node<> * node,
            const ColorShape & fill_color,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the polyline element.

Parameters

node	The node to be parsed.
fill_color	The color of the fill
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The polyline element

Definition at line 842 of file Parser.cpp.

```
845                                                 {
846     Pyline *shape = new Pyline(fill_color, stroke_color, stroke_width);
847     std::vector< Vector2Df > points = parsePoints(node);
848     for (auto point : points) {
849         shape->addPoint(point);
850     }
851     std::string fill_rule = getAttribute(node, "fill-rule");
852     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
853                     fill_rule.end());
854     shape->setFillRule(fill_rule);
855     return shape;
856 }
```

3.8.3.21 parseRect()

```
Rect * Parser::parseRect (
            rapidxml::xml_node<> * node,
            const ColorShape & fill_color,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the rect element.

Parameters

node	The node to be parsed.
fill_color	The color of the fill
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The rect element

Definition at line 784 of file Parser.cpp.

```
786                                                                                 {
787     float x = getFloatAttribute(node, "x");
788     float y = getFloatAttribute(node, "y");
789     float rx = getFloatAttribute(node, "rx");
790     float ry = getFloatAttribute(node, "ry");
791     Rect *shape =
792         new Rect(getFloatAttribute(node, "width"),
793                  getFloatAttribute(node, "height"), Vector2Df(x, y),
794                  Vector2Df(rx, ry), fill_color, stroke_color, stroke_width);
795     return shape;
796 }
```

3.8.3.22 parseShape()

```
SVGElement * Parser::parseShape (
            rapidxml::xml_node<> * node )  [private]
```

Parses the group of elements.

Parameters

node	The node to be parsed.

Returns

The group of elements

Definition at line 727 of file Parser.cpp.

```
727                                                 {
728     SVGElement *shape = NULL;
729     std::string type = node->name();
730     std::string id = "";
731     ColorShape stroke_color = parseColor(node, "stroke", id);
732     ColorShape fill_color = parseColor(node, "fill", id);
733     float stroke_width = getFloatAttribute(node, "stroke-width");
734     // Determine the type of SVG element and create the corresponding object
735     if (type == "line") {
736         shape = parseLine(node, stroke_color, stroke_width);
737     } else if (type == "rect") {
738         shape = parseRect(node, fill_color, stroke_color, stroke_width);
739     } else if (type == "circle") {
740         shape = parseCircle(node, fill_color, stroke_color, stroke_width);
741     } else if (type == "ellipse") {
742         shape = parseEllipse(node, fill_color, stroke_color, stroke_width);
743     } else if (type == "polygon") {
744         shape = parsePolygon(node, fill_color, stroke_color, stroke_width);
745     } else if (type == "polyline") {
746         shape = parsePolyline(node, fill_color, stroke_color, stroke_width);
747     } else if (type == "path") {
748         shape = parsePath(node, fill_color, stroke_color, stroke_width);
749     } else if (type == "text") {
750         shape = parseText(node, fill_color, stroke_color, stroke_width);
751     }
752
753     // Apply transformations and gradient if applicable
754     if (shape != NULL) {
755         if (type == "text") {
756             float dx = getFloatAttribute(node, "dx");
757             float dy = getFloatAttribute(node, "dy");
758             std::string transform = "translate(" + std::to_string(dx) + " " +
759                                     std::to_string(dy) + ")";
760             std::vector< std::string > transform_order =
```

```
761                 getTransformOrder(node);
762             transform_order.push_back(transform);
763             shape->setTransforms(transform_order);
764         } else
765             shape->setTransforms(getTransformOrder(node));
766         if (id != "") {
767             shape->setGradient(parseGradient(id));
768         }
769     }
770     return shape;
771 }
```

3.8.3.23 parseText()

```
Text * Parser::parseText (
            rapidxml::xml_node<> * node,
            const ColorShape & fill_color,
            const ColorShape & stroke_color,
            float stroke_width )  [private]
```

Parses the text element.

Parameters

node	The node to be parsed.
fill_color	The color of the fill
stroke_color	The color of the stroke
stroke_width	The width of the stroke

Returns

The text element

Definition at line 859 of file Parser.cpp.

```
861                                                                                 {
862     float x = getFloatAttribute(node, "x");
863     float y = getFloatAttribute(node, "y");
864     float font_size = getFloatAttribute(node, "font-size");
865     std::string text = getAttribute(node, "text");
866
867     Text *shape =
868         new Text(Vector2Df(x - (font_size * 6.6 / 40),
869                            y - font_size + (font_size * 4.4 / 40)),
870                  text, font_size, fill_color, stroke_color, stroke_width);
871
872     std::string anchor = getAttribute(node, "text-anchor");
873     anchor.erase(std::remove(anchor.begin(), anchor.end(), ' '), anchor.end());
874     shape->setAnchor(anchor);
875
876     std::string style = getAttribute(node, "font-style");
877     style.erase(std::remove(style.begin(), style.end(), ' '), style.end());
878     shape->setFontStyle(style);
879
880     return shape;
881 }
```

3.8.3.24 printShapesData()

```
void Parser::printShapesData ( )
```

Prints the data of the shapes.

Note

This function is used for debugging.

Definition at line 908 of file Parser.cpp.

```
908 { root->printData(); }
```

3.8.4 Member Data Documentation

3.8.4.1 gradients

```
std::map< std::string, Gradient* > Parser::gradients  [private]
```

The gradients of the SVG file.

Definition at line 279 of file Parser.hpp.

The documentation for this class was generated from the following files:

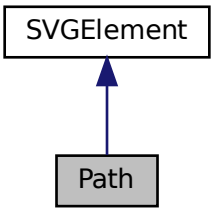
- src/Parser.hpp
- src/Parser.cpp

3.9 Path Class Reference

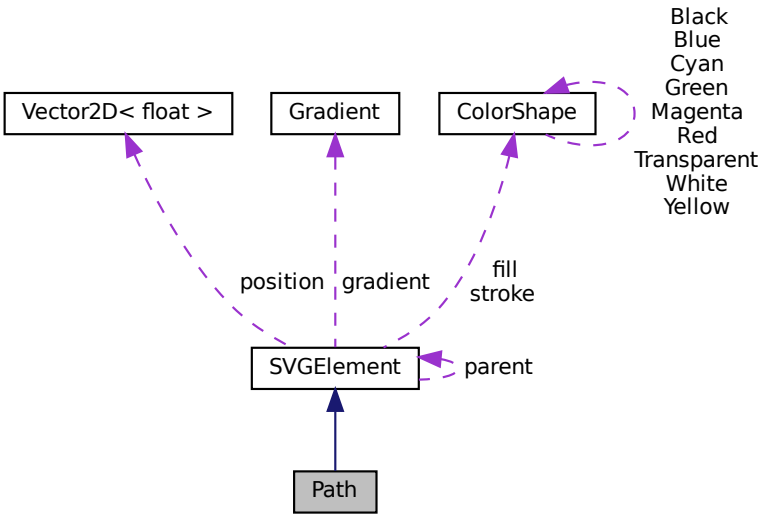
Represents a path element in 2D space.

```
#include <Path.hpp>
```

Inheritance diagram for Path:



Collaboration diagram for Path:



Public Member Functions

- Path (const ColorShape &fill, const ColorShape &stroke, float stroke_width)

  Constructs a Path object.
- std::string getClass () const override

  Gets the type of the shape.
- void addPoint (PathPoint point)

  Adds a point to the path.
- std::vector< PathPoint > getPoints () const

  Gets the vector of points in the path.
- void setFillRule (std::string fill_rule)

  Sets the fill rule of the path.
- std::string getFillRule () const

  Gets the current fill rule of the path.
- void printData () const override

  Prints the data of the shape.

Private Attributes

- std::vector< PathPoint > points

  Vector of points in the path.
- std::string fill_rule

  Fill rule of the path.

Additional Inherited Members

3.9.1 Detailed Description

Represents a path element in 2D space.

The Path class is derived from the SVGElement class and represents a path element in 2D space. The Path class is used to draw lines, curves, arcs, and other shapes. The Path class contains a vector of PathPoints that represent the points in the path.

Definition at line 28 of file Path.hpp.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Path()

```
Path::Path (
            const ColorShape & fill,
            const ColorShape & stroke,
            float stroke_width )
```

Constructs a Path object.

Parameters

fill	Fill color of the path.
stroke	Outline color of the path.
stroke_width	Thickness of the path outline.

Definition at line 3 of file Path.cpp.

```
4     : SVGElement(fill, stroke, stroke_width) {}
```

3.9.3 Member Function Documentation

3.9.3.1 addPoint()

```
void Path::addPoint (
            PathPoint point )
```

Adds a point to the path.

Parameters

point	The point to be added to the path.

Note

This function is used for adding points to the path.

Definition at line 8 of file Path.cpp.

```
8 { points.push_back(point); }
```

3.9.3.2 getClass()

```
std::string Path::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Path".

Implements SVGElement.

Definition at line 6 of file Path.cpp.

```
6 { return "Path"; }
```

3.9.3.3 getFillRule()

```
std::string Path::getFillRule ( ) const
```

Gets the current fill rule of the path.

Returns

The current fill rule of the path.

Note

The fill rule can be either "nonzero" or "evenodd".

The default fill rule is "nonzero".

Definition at line 14 of file Path.cpp.

```
14 { return fill_rule; }
```

3.9.3.4 getPoints()

```
std::vector< PathPoint > Path::getPoints ( ) const
```

Gets the vector of points in the path.

Returns

The vector of points in the path.

Definition at line 10 of file Path.cpp.

```
10 { return points; }
```

3.9.3.5 printData()

```
void Path::printData ( ) const  [override], [virtual]
```

Prints the data of the shape.

Note

This function is used for debugging purposes.

Reimplemented from SVGElement.

Definition at line 16 of file Path.cpp.

```
16                              {
17     SVGElement::printData();
18     std::cout « "Points: ";
19     for (auto point : points) {
20         std::cout « point.tc « " " « point.point.x « " " « point.point.y
21                   « " ";
22     }
23 }
```

3.9.3.6 setFillRule()

```
void Path::setFillRule (
            std::string fill_rule )
```

Sets the fill rule of the path.

Parameters

fill_rule	The new fill rule of the path.

Note

This function is used for setting the fill rule of the path.

The fill rule can be either "nonzero" or "evenodd".

Definition at line 12 of file Path.cpp.

```
12 { this->fill_rule = fill_rule; }
```

The documentation for this class was generated from the following files:

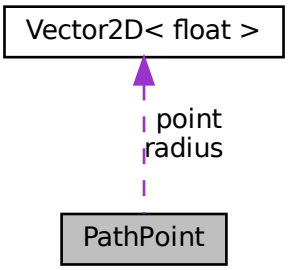
- src/graphics/Path.hpp
- src/graphics/Path.cpp

3.10 PathPoint Struct Reference

A struct that contains a point and a type of point.

`#include <Path.hpp>`

Collaboration diagram for PathPoint:



Public Attributes

- Vector2Df point
  Point in 2D space.
- char tc
  Type of point.
- Vector2Df radius {0, 0}
  Radius of the arc.
- float x_axis_rotation = 0.f
  Rotation of the arc.
- bool large_arc_flag = false
  Flag for large arc.
- bool sweep_flag = false
  Flag for sweep.

3.10.1 Detailed Description

A struct that contains a point and a type of point.

Definition at line 10 of file Path.hpp.

The documentation for this struct was generated from the following file:

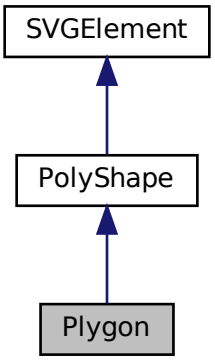
- src/graphics/Path.hpp

3.11 Plygon Class Reference

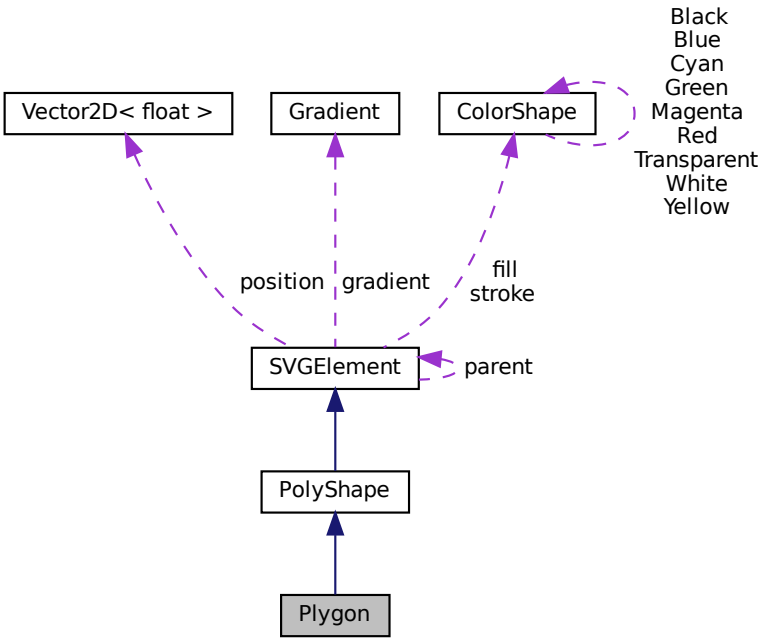
Represents a polygon in 2D space.

`#include <Polygon.hpp>`

Inheritance diagram for Plygon:



Collaboration diagram for Plygon:



Public Member Functions

- Plygon (ColorShape fill, ColorShape stroke, float stroke_width)

  Constructs a Polygon object.
- std::string getClass () const override

  Gets the type of the shape.

Additional Inherited Members

3.11.1 Detailed Description

Represents a polygon in 2D space.

The Polygon class is derived from the PolyShape class and defines a polygon with a variable number of vertices.

Definition at line 12 of file Polygon.hpp.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 Plygon()

```
Plygon::Plygon (
            ColorShape fill,
            ColorShape stroke,
            float stroke_width )
```

Constructs a Polygon object.

Parameters

fill	Fill color of the polygon (default is sf::Color::Transparent).
stroke	Outline color of the polygon (default is sf::Color::White).
stroke_width	Thickness of the polygon outline (default is 0).

Definition at line 3 of file Polygon.cpp.

```
4     : PolyShape(fill, stroke, stroke_width) {}
```

3.11.3 Member Function Documentation

3.11.3.1 getClass()

```
std::string Plygon::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Polygon".

Implements PolyShape.

Definition at line 6 of file Polygon.cpp.

```
6 { return "Polygon"; }
```

The documentation for this class was generated from the following files:

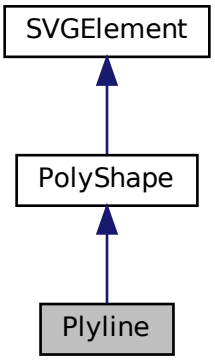
- src/graphics/Polygon.hpp
- src/graphics/Polygon.cpp

3.12 Plyline Class Reference

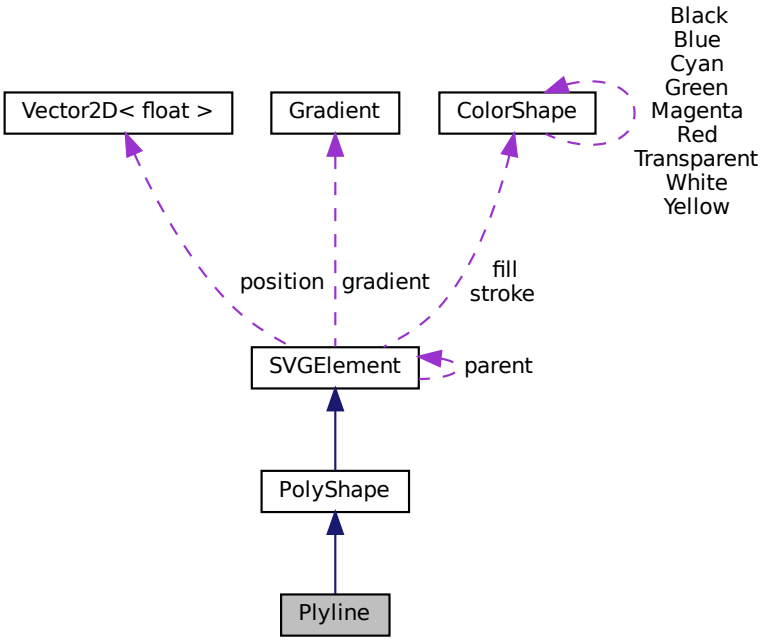
Represents a polyline in 2D space.

```
#include <Polyline.hpp>
```

Inheritance diagram for Plyline:



Collaboration diagram for Plyline:



Public Member Functions

- Plyline (const ColorShape &fill, const ColorShape &stroke, float stroke_width)

  Constructs a Polyline object.
- std::string getClass () const override

  Gets the type of the shape.

Additional Inherited Members

3.12.1 Detailed Description

Represents a polyline in 2D space.

The Polyline class is derived from the PolyShape class and defines a polyline with a variable number of vertices.

Definition at line 12 of file Polyline.hpp.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 Plyline()

```
Plyline::Plyline (
            const ColorShape & fill,
            const ColorShape & stroke,
            float stroke_width )
```

Constructs a Polyline object.

Parameters

stroke_width	The stroke width of the polyline (default is 0).
stroke	The stroke color of the polyline (default is sf::Color::White).
fill	The fill color of the polyline (default is sf::Color::Transparent).

Definition at line 3 of file Polyline.cpp.

```
5     : PolyShape(fill, stroke, stroke_width) {}
```

3.12.3 Member Function Documentation

3.12.3.1 getClass()

```
std::string Plyline::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Polyline".

Implements PolyShape.

Definition at line 7 of file Polyline.cpp.

```
7 { return "Polyline"; }
```

The documentation for this class was generated from the following files:

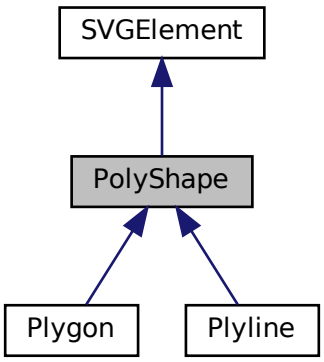
- src/graphics/Polyline.hpp
- src/graphics/Polyline.cpp

3.13 PolyShape Class Reference

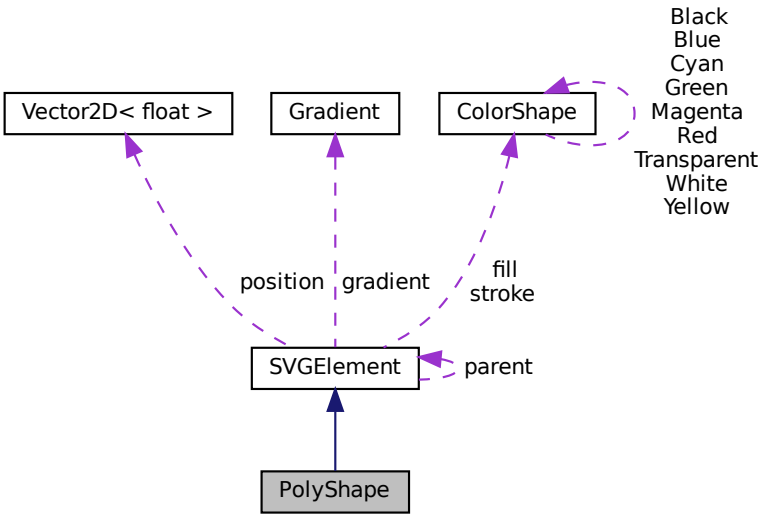
Abstract base class for polygon and polyline shapes in 2D space.

`#include <PolyShape.hpp>`

Inheritance diagram for PolyShape:



Collaboration diagram for PolyShape:



Public Member Functions

- std::string getClass () const =0

  Gets the type of the shape.
- virtual void addPoint (const Vector2Df &point)

  Adds a vertex to the shape.
- const std::vector< Vector2Df > & getPoints () const

  Gets the total number of vertices representing the shape.
- void setFillRule (std::string fill_rule)

  Sets the fill rule of the polyshape.
- std::string getFillRule () const

  Gets the fill rule of the polyshape.
- Vector2Df getMinBound () const override

  Gets the minimum bounding box of the shape.
- Vector2Df getMaxBound () const override

  Gets the maximum bounding box of the shape.
- void printData () const override

  Prints the data of the shape.

Protected Member Functions

- PolyShape (const ColorShape &fill, const ColorShape &stroke, float stroke_width)

  Constructs a PolyShape object.

Protected Attributes

- std::vector< Vector2Df > points

  Vertices of the polyshape.
- std::string fill_rule

  Fill rule of the polyshape.

3.13.1 Detailed Description

Abstract base class for polygon and polyline shapes in 2D space.

The PolyShape class is derived from the SVGElement class and defines a common interface for polyline and polygon shapes.

Definition at line 12 of file PolyShape.hpp.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 PolyShape()

```
PolyShape::PolyShape (
            const ColorShape & fill,
            const ColorShape & stroke,
            float stroke_width )  [protected]
```

Constructs a PolyShape object.

Parameters

fill	Fill color of the polyshape (default is sf::Color::Transparent).
stroke	Outline color of the polyshape (default is sf::Color::White).
stroke_width	Thickness of the polyshape outline (default is 0).

Definition at line 3 of file PolyShape.cpp.

```
5     : SVGElement(fill, stroke, stroke_width) {}
```

3.13.3 Member Function Documentation

3.13.3.1 addPoint()

```
void PolyShape::addPoint (
            const Vector2Df & point )  [virtual]
```

Adds a vertex to the shape.

Parameters

point	The position of the vertex to be added.

Definition at line 7 of file PolyShape.cpp.

```
7 { points.push_back(point); }
```

3.13.3.2 getClass()

```
std::string PolyShape::getClass ( ) const  [pure virtual]
```

Gets the type of the shape.

Note

This function is pure virtual and must be implemented by derived classes.

Implements SVGElement.

Implemented in Plyline, and Plygon.

3.13.3.3 getFillRule()

```
std::string PolyShape::getFillRule ( ) const
```

Gets the fill rule of the polyshape.

Returns

The fill rule of the polyshape.

Definition at line 15 of file PolyShape.cpp.

```
15 { return fill_rule; }
```

3.13.3.4 getMaxBound()

```
Vector2Df PolyShape::getMaxBound ( ) const  [override], [virtual]
```

Gets the maximum bounding box of the shape.

Returns

The maximum bounding box of the shape.

Reimplemented from SVGElement.

Definition at line 27 of file PolyShape.cpp.

```
27                                          {
28     float max_x = points[0].x;
29     float max_y = points[0].y;
30     for (auto& point : points) {
31         max_x = std::max(max_x, point.x);
32         max_y = std::max(max_y, point.y);
33     }
34     return Vector2Df(max_x, max_y);
35 }
```

3.13.3.5 getMinBound()

```
Vector2Df PolyShape::getMinBound ( ) const  [override], [virtual]
```

Gets the minimum bounding box of the shape.

Returns

The minimum bounding box of the shape.

Reimplemented from SVGElement.

Definition at line 17 of file PolyShape.cpp.

```
17                                          {
18     float min_x = points[0].x;
19     float min_y = points[0].y;
20     for (auto& point : points) {
21         min_x = std::min(min_x, point.x);
22         min_y = std::min(min_y, point.y);
23     }
24     return Vector2Df(min_x, min_y);
25 }
```

3.13.3.6 getPoints()

```
const std::vector< Vector2Df > & PolyShape::getPoints ( ) const
```

Gets the total number of vertices representing the shape.

Returns

The number of vertices representing the shape.

Definition at line 9 of file PolyShape.cpp.

```
9 { return points; }
```

3.13.3.7 printData()

```
void PolyShape::printData ( ) const  [override], [virtual]
```

Prints the data of the shape.

Note

This function is used for debugging purposes.

Reimplemented from SVGElement.

Definition at line 37 of file PolyShape.cpp.

```
37                                  {
38     SVGElement::printData();
39     std::cout « "Points: ";
40     for (auto& point : getPoints()) {
41         std::cout « point.x « "," « point.y « " ";
42     }
43     std::cout « std::endl;
44 }
```

3.13.3.8 setFillRule()

```
void PolyShape::setFillRule (
            std::string fill_rule )
```

Sets the fill rule of the polyshape.

Parameters

fill_rule	The new fill rule of the polyshape.

Definition at line 11 of file PolyShape.cpp.

```
11                                                  {
12     this->fill_rule = fill_rule;
13 }
```

The documentation for this class was generated from the following files:

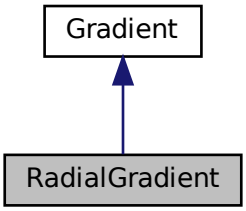
- src/graphics/PolyShape.hpp
- src/graphics/PolyShape.cpp

3.14 RadialGradient Class Reference

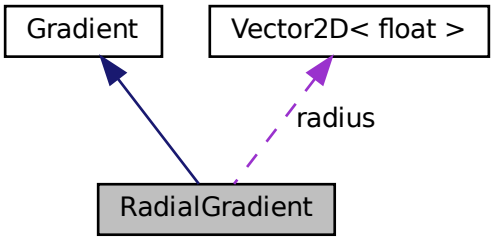
A class that represents a radial gradient.

`#include <RadialGradient.hpp>`

Inheritance diagram for RadialGradient:



Collaboration diagram for RadialGradient:



Public Member Functions

- RadialGradient (std::vector< Stop > stops, std::pair< Vector2Df, Vector2Df > points, Vector2Df radius, std::string units)

  Constructs a RadialGradient object.
- std::string getClass () const override

  Gets the type of the gradient.
- Vector2Df getRadius () const

  Gets the radius of the gradient.

Private Attributes

- Vector2Df radius

  The radius of the gradient.

3.14.1 Detailed Description

A class that represents a radial gradient.

The RadialGradient class is derived from the Gradient class and represents a radial gradient. It contains a vector of Stop objects that represent the the stops of the gradient. It also contains a pair of Vector2D objects that represent the start and end points of the gradient.

Definition at line 14 of file RadialGradient.hpp.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 RadialGradient()

```
RadialGradient::RadialGradient (
            std::vector< Stop > stops,
            std::pair< Vector2Df, Vector2Df > points,
            Vector2Df radius,
            std::string units )
```

Constructs a RadialGradient object.

Parameters

stops	The stops of the gradient.
points	The start and end points of the gradient.
radius	The radius of the gradient.
units	The units of the gradient.

Definition at line 3 of file RadialGradient.cpp.

```
6     : Gradient(stops, points, units) {
7     this->radius = radius;
8 }
```

3.14.3 Member Function Documentation

3.14.3.1 getClass()

```
std::string RadialGradient::getClass ( ) const  [override], [virtual]
```

Gets the type of the gradient.

Returns

The string "RadialGradient".

Note

This function is used for determining the type of the gradient.

Implements Gradient.

Definition at line 10 of file RadialGradient.cpp.

```
10 { return "RadialGradient"; }
```

3.14.3.2 getRadius()

```
Vector2Df RadialGradient::getRadius ( ) const
```

Gets the radius of the gradient.

Returns

The radius of the gradient.

Definition at line 12 of file RadialGradient.cpp.

```
12 { return radius; }
```

The documentation for this class was generated from the following files:

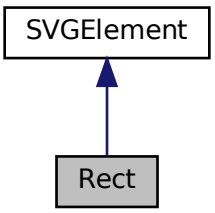
- src/graphics/RadialGradient.hpp
- src/graphics/RadialGradient.cpp

3.15 Rect Class Reference

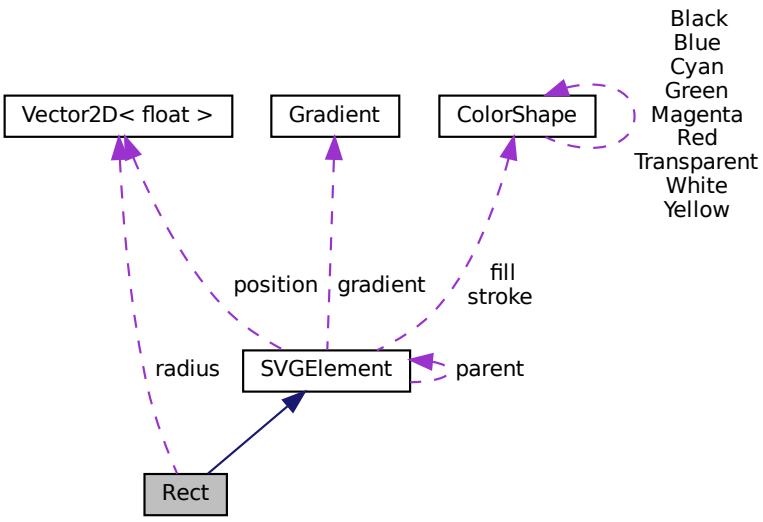
Represents a rectangle in 2D space.

`#include <Rect.hpp>`

Inheritance diagram for Rect:



Collaboration diagram for Rect:



Public Member Functions

- Rect (float width, float height, Vector2Df position, Vector2Df radius, const ColorShape &fill, const ColorShape &stroke, float stroke_width)

  Constructs a Rect object.
- std::string getClass () const override

Gets the type of the shape.

- void setWidth (float width)

  Sets the width of the rectangle.
- float getWidth () const

  Gets the width of the rectangle.
- void setHeight (float height)

  Sets the height of the rectangle.
- float getHeight () const

  Gets the height of the rectangle.
- void setRadius (const Vector2Df &radius)

  Sets the radii of the rectangle.
- Vector2Df getRadius () const

  Gets the radii of the rectangle.
- void printData () const override

  Prints the data of the rectangle.

Private Attributes

- float width

  Width of the rectangle.
- float height

  Height of the rectangle.
- Vector2Df radius

  Radii of the rectangle in the x and y directions.

Additional Inherited Members

3.15.1 Detailed Description

Represents a rectangle in 2D space.

The Rect class is derived from the SVGElement class and defines a rectangle with a specified width, height, position, fill color, stroke color, and stroke thickness.

Definition at line 13 of file Rect.hpp.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 Rect()

```
Rect::Rect (
            float width,
            float height,
            Vector2Df position,
            Vector2Df radius,
            const ColorShape & fill,
            const ColorShape & stroke,
            float stroke_width )
```

Constructs a Rect object.

Parameters

width	The width of the rectangle.
height	The height of the rectangle.
position	The position of the rectangle.
radius	The radii of the rectangle in the x and y directions.
fill	Fill color of the rectangle.
stroke	Outline color of the rectangle.
stroke_width	Thickness of the rectangle outline.

Definition at line 3 of file Rect.cpp.

```
5     : SVGElement(fill, stroke, stroke_width, position), width(width),
6       height(height), radius(radius) {}
```

3.15.3 Member Function Documentation

3.15.3.1 getClass()

```
std::string Rect::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Rect".

Implements SVGElement.

Definition at line 8 of file Rect.cpp.

```
8 { return "Rect"; }
```

3.15.3.2 getHeight()

```
float Rect::getHeight ( ) const
```

Gets the height of the rectangle.

Returns

The height of the rectangle.

Definition at line 16 of file Rect.cpp.

```
16 { return height; }
```

3.15.3.3 getRadius()

```
Vector2Df Rect::getRadius ( ) const
```

Gets the radii of the rectangle.

Returns

The radii of the rectangle.

Definition at line 20 of file Rect.cpp.

```
20 { return radius; }
```

3.15.3.4 getWidth()

```
float Rect::getWidth ( ) const
```

Gets the width of the rectangle.

Returns

The width of the rectangle.

Definition at line 12 of file Rect.cpp.

```
12 { return width; }
```

3.15.3.5 printData()

```
void Rect::printData ( ) const  [override], [virtual]
```

Prints the data of the rectangle.

Note

This function is used for debugging purposes.

Reimplemented from SVGElement.

Definition at line 22 of file Rect.cpp.

```
22                              {
23     SVGElement::printData();
24     std::cout « "Width: " « getWidth() « std::endl;
25     std::cout « "Height: " « getHeight() « std::endl;
26     std::cout « "Radius: " « getRadius().x « " " « getRadius().y
27               « std::endl;
28 }
```

3.15.3.6 setHeight()

```
void Rect::setHeight (
             float height )
```

Sets the height of the rectangle.

Parameters

height	The new height of the rectangle.

Definition at line 14 of file Rect.cpp.

```
14 { this->height = height; }
```

3.15.3.7 setRadius()

```
void Rect::setRadius (
            const Vector2Df & radius )
```

Sets the radii of the rectangle.

Parameters

radius	The new radii of the rectangle.

Definition at line 18 of file Rect.cpp.

```
18 { this->radius = radius; }
```

3.15.3.8 setWidth()

```
void Rect::setWidth (
            float width )
```

Sets the width of the rectangle.

Parameters

width	The new width of the rectangle.

Definition at line 10 of file Rect.cpp.

```
10 { this->width = width; }
```

The documentation for this class was generated from the following files:

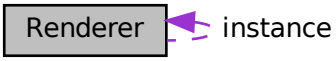
- src/graphics/Rect.hpp
- src/graphics/Rect.cpp

3.16 Renderer Class Reference

Singleton class responsible for rendering shapes using GDI+.

`#include <Renderer.hpp>`

Collaboration diagram for Renderer:



Public Member Functions

- Renderer (const Renderer &)=delete

  Deleted copy constructor to enforce the singleton pattern.
- void operator= (const Renderer &)=delete

  Deleted copy assignment operator to enforce the singleton pattern.
- void draw (Gdiplus::Graphics &graphics, Group *group) const

  Draws a shape using Gdiplus::Graphics based on its type.

Static Public Member Functions

- static Renderer * getInstance ()

  Gets the singleton instance of the Renderer class.

Private Member Functions

- void applyTransform (std::vector< std::string > transform_order, Gdiplus::Graphics &graphics) const

  Utility function to apply a series of transformations to the graphics context.
- void drawLine (Gdiplus::Graphics &graphics, Line *line) const

  Draws a line shape using Gdiplus::Graphics.
- void drawRectangle (Gdiplus::Graphics &graphics, Rect *rectangle) const

  Draws a rectangle shape using Gdiplus::Graphics.
- void drawCircle (Gdiplus::Graphics &graphics, Circle *circle) const

  Draws a circle shape using Gdiplus::Graphics.
- void drawEllipse (Gdiplus::Graphics &graphics, Ell *ellipse) const

  Draws an ellipse shape using Gdiplus::Graphics.
- void drawPolygon (Gdiplus::Graphics &graphics, Plygon *polygon) const

  Draws a polygon shape using Gdiplus::Graphics.
- void drawText (Gdiplus::Graphics &graphics, Text *text) const

  Draws text using Gdiplus::Graphics.
- void drawPolyline (Gdiplus::Graphics &graphics, Plyline *polyline) const

  Draws a polyline shape using Gdiplus::Graphics.
- void drawPath (Gdiplus::Graphics &graphics, Path *path) const

  Draws a path shape using Gdiplus::Graphics.
- Gdiplus::Brush * getBrush (SVGElement *shape, Gdiplus::RectF bound) const

  Gets the Gdiplus::brush object for the shape fill.

- void applyTransformsOnBrush (std::vector< std::string > transform_order, Gdiplus::LinearGradientBrush *&brush) const

  Utility function to apply a series of transformations to the brush object.
- void applyTransformsOnBrush (std::vector< std::string > transform_order, Gdiplus::PathGradientBrush *&brush) const

  Utility function to apply a series of transformations to the brush object.
- Renderer ()

  Private constructor for the Renderer class.

Static Private Attributes

- static Renderer * instance = nullptr

  Singleton instance of the Renderer class.

3.16.1 Detailed Description

Singleton class responsible for rendering shapes using GDI+.

The Renderer class provides a singleton instance for drawing SVGElement-based shapes using Gdiplus::Graphics. It supports various shapes such as lines, rectangles, circles, ellipses, text, polygons, polylines, and paths. The shapes are drawn in a polymorphic manner using the draw function, which takes a Gdiplus::Graphics context and an SVGElement. The draw function dynamically determines the type of the shape and invokes the corresponding draw method to render the shape with all necessary details. The detailed information for each shape is obtained from an SVG file and processed through the draw function in a polymorphic way.

Definition at line 24 of file Renderer.hpp.

3.16.2 Member Function Documentation

3.16.2.1 applyTransform()

```
void Renderer::applyTransform (
            std::vector< std::string > transform_order,
            Gdiplus::Graphics & graphics ) const  [private]
```

Utility function to apply a series of transformations to the graphics context.

Parameters

transform_order	The order in which transformations should be applied.
graphics	The Gdiplus::Graphics context to apply transformations to.

Definition at line 55 of file Renderer.cpp.

```
56                                                                  {
57     for (auto type : transform_order) {
58         if (type.find("translate") != std::string::npos) {
59             float trans_x = getTranslate(type).first,
```

```
60                  trans_y = getTranslate(type).second;
61              graphics.TranslateTransform(trans_x, trans_y);
62          } else if (type.find("rotate") != std::string::npos) {
63              float degree = getRotate(type);
64              graphics.RotateTransform(degree);
65          } else if (type.find("scale") != std::string::npos) {
66              if (type.find(",") != std::string::npos) {
67                  float scale_x = getScaleXY(type).first,
68                        scale_y = getScaleXY(type).second;
69                  graphics.ScaleTransform(scale_x, scale_y);
70              } else {
71                  float scale = getScale(type);
72                  graphics.ScaleTransform(scale, scale);
73              }
74          }
75      }
76 }
```

3.16.2.2 applyTransformsOnBrush() [1/2]

```
void Renderer::applyTransformsOnBrush (
            std::vector< std::string > transform_order,
            Gdiplus::LinearGradientBrush *& brush ) const  [private]
```

Utility function to apply a series of transformations to the brush object.

Parameters

transform_order	The order in which transformations should be applied.
brush	The Gdiplus::LinearGradientBrush object for the shape fill.

Definition at line 791 of file Renderer.cpp.

```
793                                                 {
794     for (auto type : transform_order) {
795         if (type.find("translate") != std::string::npos) {
796             // Apply translation transformation
797             float trans_x = getTranslate(type).first,
798                   trans_y = getTranslate(type).second;
799             brush->TranslateTransform(trans_x, trans_y);
800         } else if (type.find("rotate") != std::string::npos) {
801             // Apply rotation transformation
802             float degree = getRotate(type);
803             brush->RotateTranform(degree);
804         } else if (type.find("scale") != std::string::npos) {
805             // Apply scaling transformation
806             if (type.find(",") != std::string::npos) {
807                 float scale_x = getScaleXY(type).first,
808                       scale_y = getScaleXY(type).second;
809                 brush->ScaleTransform(scale_x, scale_y);
810             } else {
811                 float scale = getScale(type);
812                 brush->ScaleTransform(scale, scale);
813             }
814         } else if (type.find("matrix") != std::string::npos) {
815             // Apply matrix transformation
816             float a = 0, b = 0, c = 0, d = 0, e = 0, f = 0;
817             if (type.find(",") != std::string::npos) {
818                 type.erase(std::remove(type.begin(), type.end(), ','),
819                            type.end());
820             }
821             sscanf(type.c_str(), "matrix(%f %f %f %f %f %f)", &a, &b, &c, &d,
822                    &e, &f);
823             Gdiplus::Matrix matrix(a, b, c, d, e, f);
824             brush->SetTransform(&matrix);
825         }
826     }
827 }
```

3.16.2.3 applyTransformsOnBrush() [2/2]

```
void Renderer::applyTransformsOnBrush (
            std::vector< std::string > transform_order,
            Gdiplus::PathGradientBrush *& brush ) const  [private]
```

Utility function to apply a series of transformations to the brush object.

Parameters

transform_order	The order in which transformations should be applied.
brush	The Gdiplus::PathGradientBrush object for the shape fill.

Definition at line 830 of file Renderer.cpp.

```
832                                                 {
833     for (auto type : transform_order) {
834         if (type.find("translate") != std::string::npos) {
835             float trans_x = getTranslate(type).first,
836                   trans_y = getTranslate(type).second;
837             brush->TranslateTransform(trans_x, trans_y);
838         } else if (type.find("rotate") != std::string::npos) {
839             float degree = getRotate(type);
840             brush->RotateTransform(degree);
841         } else if (type.find("scale") != std::string::npos) {
842             if (type.find(",") != std::string::npos) {
843                 float scale_x = getScaleXY(type).first,
844                       scale_y = getScaleXY(type).second;
845                 brush->ScaleTransform(scale_x, scale_y);
846             } else {
847                 float scale = getScale(type);
848                 brush->ScaleTransform(scale, scale);
849             }
850         } else if (type.find("matrix") != std::string::npos) {
851             float a = 0, b = 0, c = 0, d = 0, e = 0, f = 0;
852             if (type.find(",") != std::string::npos) {
853                 type.erase(std::remove(type.begin(), type.end(), ','),
854                            type.end());
855             }
856             sscanf(type.c_str(), "matrix(%f %f %f %f %f %f)", &a, &b, &c, &d,
857                    &e, &f);
858             Gdiplus::Matrix matrix(a, b, c, d, e, f);
859             brush->SetTransform(&matrix);
860         }
861     }
862 }
```

3.16.2.4 draw()

```
void Renderer::draw (
            Gdiplus::Graphics & graphics,
            Group * group ) const
```

Draws a shape using Gdiplus::Graphics based on its type.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
shape	The SVGElement representing the shape to be drawn.

Definition at line 79 of file Renderer.cpp.

```
79                                                  {
```

```
80      for (auto shape : group->getElements()) {
81          // Store the original transformation matrix
82          Gdiplus::Matrix original;
83          graphics.GetTransform(&original);
84
85          // Apply the transformations for the current shape
86          applyTransform(shape->getTransforms(), graphics);
87
88          // Draw the specific shape based on its class
89          if (shape->getClass() == "Group") {
90              Group* group = dynamic_cast< Group* >(shape);
91              draw(graphics, group);
92          } else if (shape->getClass() == "Polyline") {
93              Plyline* polyline = dynamic_cast< Plyline* >(shape);
94              drawPolyline(graphics, polyline);
95          } else if (shape->getClass() == "Text") {
96              Text* text = dynamic_cast< Text* >(shape);
97              drawText(graphics, text);
98          } else if (shape->getClass() == "Rect") {
99              Rect* rectangle = dynamic_cast< Rect* >(shape);
100             drawRectangle(graphics, rectangle);
101         } else if (shape->getClass() == "Circle") {
102             Circle* circle = dynamic_cast< Circle* >(shape);
103             drawCircle(graphics, circle);
104         } else if (shape->getClass() == "Ellipse") {
105             Ell* ellipse = dynamic_cast< Ell* >(shape);
106             drawEllipse(graphics, ellipse);
107         } else if (shape->getClass() == "Line") {
108             Line* line = dynamic_cast< Line* >(shape);
109             drawLine(graphics, line);
110         } else if (shape->getClass() == "Polygon") {
111             Plygon* polygon = dynamic_cast< Plygon* >(shape);
112             drawPolygon(graphics, polygon);
113         } else if (shape->getClass() == "Path") {
114             Path* path = dynamic_cast< Path* >(shape);
115             drawPath(graphics, path);
116         }
117         graphics.SetTransform(&original);
118     }
119 }
```

3.16.2.5 drawCircle()

```
void Renderer::drawCircle (
            Gdiplus::Graphics & graphics,
            Circle * circle ) const  [private]
```

Draws a circle shape using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
circle	The Circle object representing the circle to be drawn.

Definition at line 192 of file Renderer.cpp.

```
192                                                                     {
193     ColorShape outline_color = circle->getOutlineColor();
194     Gdiplus::Pen circle_outline(
195         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
196                        outline_color.b),
197         circle->getOutlineThickness());
198
199     // Create a bounding rectangle for the circle
200     Vector2Df min_bound = circle->getMinBound();
201     Vector2Df max_bound = circle->getMaxBound();
202     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
203                          max_bound.y - min_bound.y);
204     Gdiplus::Brush* circle_fill = getBrush(circle, bound);
205
206     // Check if the circle has a gradient fill
207     if (Gdiplus::PathGradientBrush* brush =
```

```
208             dynamic_cast< Gdiplus::PathGradientBrush* >(circle_fill)) {
209         ColorShape color = circle->getGradient()->getStops().back().getColor();
210         Gdiplus::SolidBrush corner_fill(
211             Gdiplus::Color(color.a, color.r, color.g, color.b));
212         graphics.FillEllipse(
213             &corner_fill, circle->getPosition().x - circle->getRadius().x,
214             circle->getPosition().y - circle->getRadius().y,
215             circle->getRadius().x * 2, circle->getRadius().y * 2);
216     }
217
218     graphics.FillEllipse(circle_fill,
219                          circle->getPosition().x - circle->getRadius().x,
220                          circle->getPosition().y - circle->getRadius().y,
221                          circle->getRadius().x * 2, circle->getRadius().y * 2);
222     graphics.DrawEllipse(&circle_outline,
223                          circle->getPosition().x - circle->getRadius().x,
224                          circle->getPosition().y - circle->getRadius().y,
225                          circle->getRadius().x * 2, circle->getRadius().x * 2);
226
227     delete circle_fill;
228 }
```

3.16.2.6 drawEllipse()

```
void Renderer::drawEllipse (
            Gdiplus::Graphics & graphics,
            Ell * ellipse ) const  [private]
```

Draws an ellipse shape using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
ellipse	The Ell object representing the ellipse to be drawn.

Definition at line 231 of file Renderer.cpp.

```
231                                                                         {
232     ColorShape outline_color = ellipse->getOutlineColor();
233
234     Gdiplus::Pen ellipse_outline(
235         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
236                        outline_color.b),
237         ellipse->getOutlineThickness());
238
239     // Create a bounding rectangle for the ellipse
240     Vector2Df min_bound = ellipse->getMinBound();
241     Vector2Df max_bound = ellipse->getMaxBound();
242     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
243                          max_bound.y - min_bound.y);
244     Gdiplus::Brush* ellipse_fill = getBrush(ellipse, bound);
245
246     if (Gdiplus::PathGradientBrush* brush =
247             dynamic_cast< Gdiplus::PathGradientBrush* >(ellipse_fill)) {
248         ColorShape color = ellipse->getGradient()->getStops().back().getColor();
249         Gdiplus::SolidBrush corner_fill(
250             Gdiplus::Color(color.a, color.r, color.g, color.b));
251         graphics.FillEllipse(
252             &corner_fill, ellipse->getPosition().x - ellipse->getRadius().x,
253             ellipse->getPosition().y - ellipse->getRadius().y,
254             ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
255     }
256
257     graphics.FillEllipse(
258         ellipse_fill, ellipse->getPosition().x - ellipse->getRadius().x,
259         ellipse->getPosition().y - ellipse->getRadius().y,
260         ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
261     graphics.DrawEllipse(
262         &ellipse_outline, ellipse->getPosition().x - ellipse->getRadius().x,
263         ellipse->getPosition().y - ellipse->getRadius().y,
264         ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
265
```

```
266     delete ellipse_fill;
267 }
```

3.16.2.7 drawLine()

```
void Renderer::drawLine (
            Gdiplus::Graphics & graphics,
            Line * line ) const  [private]
```

Draws a line shape using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
line	The Line object representing the line to be drawn.

Definition at line 122 of file Renderer.cpp.

```
122                                                                 {
123     // Extract color and thickness information from the Line object
124     ColorShape color = line->getOutlineColor();
125     Gdiplus::Pen linePen(Gdiplus::Color(color.a, color.r, color.g, color.b),
126                          line->getOutlineThickness());
127     // Extract start and end points from the Line object
128     Gdiplus::PointF startPoint(line->getPosition().x, line->getPosition().y);
129     Gdiplus::PointF endPoint(line->getDirection().x, line->getDirection().y);
130     graphics.DrawLine(&linePen, startPoint, endPoint);
131 }
```

3.16.2.8 drawPath()

```
void Renderer::drawPath (
            Gdiplus::Graphics & graphics,
            Path * path ) const  [private]
```

Draws a path shape using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
path	The Path object representing the path to be drawn.

Definition at line 438 of file Renderer.cpp.

```
438                                                                 {
439     ColorShape outline_color = path->getOutlineColor();
440     Gdiplus::Pen path_outline(Gdiplus::Color(outline_color.a, outline_color.r,
441                                              outline_color.g, outline_color.b),
442                               path->getOutlineThickness());
443
444     // Fill the path by rules
445     Gdiplus::FillMode fill_mode;
446     if (path->getFillRule() == "evenodd") {
447         fill_mode = Gdiplus::FillModeAlternate;
448     } else if (path->getFillRule() == "nonzero") {
449         fill_mode = Gdiplus::FillModeWinding;
450     }
```

```
451     Gdiplus::GraphicsPath gdi_path(fill_mode);
452
453     const std::vector< PathPoint >& points = path->getPoints();
454     int n = points.size();
455     Vector2Df first_point{0, 0}, cur_point{0, 0};
456
457     // Construct the path
458     for (int i = 0; i < n; ++i) {
459         if (points[i].tc == 'm') {
460             // If the command is m, then start a new figure
461             first_point = points[i].point;
462             gdi_path.StartFigure();
463             cur_point = first_point;
464         } else if (points[i].tc == 'l' || points[i].tc == 'h' ||
465                    points[i].tc == 'v') {
466             // If the command is l, h, or v, then add a line to the path
467             gdi_path.AddLine(cur_point.x, cur_point.y, points[i].point.x,
468                              points[i].point.y);
469             cur_point = points[i].point;
470         } else if (points[i].tc == 'c') {
471             // If the command is c, then add a bezier curve to the path
472             if (i + 2 < n) {
473                 Vector2Df control_point1 = points[i].point;
474                 Vector2Df control_point2 = points[i + 1].point;
475                 Vector2Df control_point3 = points[i + 2].point;
476                 gdi_path.AddBezier(cur_point.x, cur_point.y, control_point1.x,
477                                    control_point1.y, control_point2.x,
478                                    control_point2.y, control_point3.x,
479                                    control_point3.y);
480                 i += 2;
481                 cur_point = control_point3;
482             }
483         } else if (points[i].tc == 'z') {
484             // If the command is z, then close the figure
485             gdi_path.CloseFigure();
486             cur_point = first_point;
487         } else if (points[i].tc == 's') {
488             // If the command is s, then add a bezier curve to the path
489             if (i + 1 < n) {
490                 // Calculate the first control point
491                 Vector2Df auto_control_point;
492                 if (i > 0 &&
493                     (points[i - 1].tc == 'c' || points[i - 1].tc == 's')) {
494                     auto_control_point.x =
495                         cur_point.x * 2 - points[i - 2].point.x;
496                     auto_control_point.y =
497                         cur_point.y * 2 - points[i - 2].point.y;
498                 } else {
499                     auto_control_point = cur_point;
500                 }
501                 // Calculate the rest control points
502                 Vector2Df control_point2 = points[i].point;
503                 Vector2Df control_point3 = points[i + 1].point;
504                 gdi_path.AddBezier(cur_point.x, cur_point.y,
505                                    auto_control_point.x, auto_control_point.y,
506                                    control_point2.x, control_point2.y,
507                                    control_point3.x, control_point3.y);
508                 i += 1;
509                 cur_point = control_point3;
510             }
511         } else if (points[i].tc == 'q') {
512             // If the command is q, then add a quadratic bezier curve to the
513             if (i + 1 < n) {
514                 // Calculate the control point and its end point
515                 Vector2Df control_point = points[i].point;
516                 Vector2Df end_point = points[i + 1].point;
517
518                 // Add the curve to the path
519                 Gdiplus::PointF q_points[3];
520                 q_points[0] = Gdiplus::PointF{cur_point.x, cur_point.y};
521                 q_points[1] = Gdiplus::PointF{control_point.x, control_point.y};
522                 q_points[2] = Gdiplus::PointF{end_point.x, end_point.y};
523                 gdi_path.AddCurve(q_points, 3);
524                 cur_point = points[i + 1].point;
525                 i += 1;
526             }
527         } else if (points[i].tc == 't') {
528             // Calculate reflection control point
529             Vector2Df auto_control_point;
530             if (i > 0 && (points[i - 1].tc == 'q' || points[i - 1].tc == 't')) {
531                 // If the previous point is a quadratic bezier or a smooth
532                 // quadratic bezier,
533                 // calculate the reflection control point using the reflection
534                 // formula
535                 auto_control_point.x = cur_point.x * 2 - points[i - 2].point.x;
536                 auto_control_point.y = cur_point.y * 2 - points[i - 2].point.y;
537             } else {
```

```
538                 // Otherwise, use the current point as the control point
539                 auto_control_point = cur_point;
540             }
541             Vector2Df end_point = points[i].point;
542             Gdiplus::PointF t_points[3];
543             t_points[0] = Gdiplus::PointF{cur_point.x, cur_point.y};
544             t_points[1] =
545                 Gdiplus::PointF{auto_control_point.x, auto_control_point.y};
546             t_points[2] = Gdiplus::PointF{end_point.x, end_point.y};
547             // Add the cubic bezier curve to the path
548             gdi_path.AddCurve(t_points, 3);
549             cur_point = points[i].point;
550         } else if (points[i].tc == 'a') {
551             float rx = points[i].radius.x;
552             float ry = points[i].radius.y;
553             // If either radius is zero, treat it as a line segment
554             if (rx == 0 || ry == 0) {
555                 gdi_path.AddLine(cur_point.x, cur_point.y, points[i].point.x,
556                                  points[i].point.y);
557                 cur_point = points[i].point;
558                 continue;
559             }
560             if (rx < 0) {
561                 rx = std::fabs(rx);
562             }
563             if (ry < 0) {
564                 ry = std::fabs(ry);
565             }
566
567             float x_axis_rotation = points[i].x_axis_rotation;
568             bool large_arc_flag = points[i].large_arc_flag;
569             bool sweep_flag = points[i].sweep_flag;
570             Vector2Df end_point{points[i].point.x, points[i].point.y};
571             // Calculate angles and points for the elliptical arc
572             float angle = x_axis_rotation * acos(-1) / 180.0;
573             float cosAngle = cos(angle);
574             float sinAngle = sin(angle);
575
576             Vector2Df point1;
577             float X = (cur_point.x - end_point.x) / 2.0;
578             float Y = (cur_point.y - end_point.y) / 2.0;
579             point1.x = (cosAngle * cosAngle + sinAngle * sinAngle) * X;
580             point1.y = (cosAngle * cosAngle + sinAngle * sinAngle) * Y;
581             // Correction of out-of-range radii
582             float radii_check = (point1.x * point1.x) / (rx * rx) +
583                                 (point1.y * point1.y) / (ry * ry);
584             if (radii_check > 1.0) {
585                 rx = std::sqrt(radii_check) * rx;
586                 ry = std::sqrt(radii_check) * ry;
587             }
588
589             float sign = (large_arc_flag == sweep_flag ? -1.0 : 1.0);
590             Vector2Df point2;
591             float numo = (rx * rx) * (ry * ry) -
592                          (rx * rx) * (point1.y * point1.y) -
593                          (ry * ry) * (point1.x * point1.x);
594             float deno = (rx * rx) * (point1.y * point1.y) +
595                          (ry * ry) * (point1.x * point1.x);
596
597             if (numo < 0) {
598                 numo = std::fabs(numo);
599             }
600
601             point2.x = sign * std::sqrt(numo / deno) * ((rx * point1.y) / ry);
602             point2.y = sign * std::sqrt(numo / deno) * ((-ry * point1.x) / rx);
603
604             Vector2Df center;
605             X = (cur_point.x + end_point.x) / 2.0;
606             Y = (cur_point.y + end_point.y) / 2.0;
607             center.x =
608                 (cosAngle * cosAngle + sinAngle * sinAngle) * point2.x + X;
609             center.y =
610                 (cosAngle * cosAngle + sinAngle * sinAngle) * point2.y + Y;
611
612             float start_angle =
613                 atan2((point1.y - point2.y) / ry, (point1.x - point2.x) / rx);
614             float end_angle =
615                 atan2((-point1.y - point2.y) / ry, (-point1.x - point2.x) / rx);
616
617             float delta_angle = end_angle - start_angle;
618
619             if (sweep_flag && delta_angle < 0) {
620                 delta_angle += 2.0 * acos(-1);
621             } else if (!sweep_flag && delta_angle > 0) {
622                 delta_angle -= 2.0 * acos(-1);
623             }
624
```

```
625             float start_angle_degree =
626                 std::fmod((start_angle * 180.0) / acos(-1), 360);
627             float delta_angle_degree =
628                 std::fmod((delta_angle * 180.0) / acos(-1), 360);
629             // Add the elliptical arc to the path
630             gdi_path.AddArc(center.x - rx, center.y - ry, 2.0 * rx, 2.0 * ry,
631                             start_angle_degree, delta_angle_degree);
632
633             cur_point = end_point;
634         }
635     }
636
637     Gdiplus::RectF bound;
638     gdi_path.GetBounds(&bound);
639     Gdiplus::Brush* path_fill = getBrush(path, bound);
640     Gdiplus::Region region(&gdi_path);
641
642     if (Gdiplus::PathGradientBrush* brush =
643             dynamic_cast< Gdiplus::PathGradientBrush* >(path_fill)) {
644         ColorShape color = path->getGradient()->getStops().back().getColor();
645         Gdiplus::SolidBrush corner_fill(
646             Gdiplus::Color(color.a, color.r, color.g, color.b));
647
648         if (path->getGradient()->getUnits() == "userSpaceOnUse") {
649             float cx = path->getGradient()->getPoints().first.x;
650             float cy = path->getGradient()->getPoints().first.y;
651             float r = dynamic_cast< RadialGradient* >(path->getGradient())
652                           ->getRadius()
653                           .x;
654             Gdiplus::GraphicsPath fill_path(fill_mode);
655             fill_path.AddEllipse(cx - r, cy - r, 2 * r, 2 * r);
656
657             for (auto type : path->getGradient()->getTransforms()) {
658                 if (type.find("matrix") != std::string::npos) {
659                     float a = 0, b = 0, c = 0, d = 0, e = 0, f = 0;
660                     if (type.find(",") != std::string::npos) {
661                         type.erase(std::remove(type.begin(), type.end(), ','),
662                                    type.end());
663                     }
664                     sscanf(type.c_str(), "matrix(%f %f %f %f %f %f)", &a, &b,
665                            &c, &d, &e, &f);
666                     Gdiplus::Matrix matrix(a, b, c, d, e, f);
667                     fill_path.Transform(&matrix);
668                 }
669             }
670             region.Exclude(&fill_path);
671         }
672         graphics.FillRegion(&corner_fill, &region);
673     }
674
675     graphics.FillPath(path_fill, &gdi_path);
676     graphics.DrawPath(&path_outline, &gdi_path);
677
678     delete path_fill;
679 }
```

3.16.2.9 drawPolygon()

```
void Renderer::drawPolygon (
            Gdiplus::Graphics & graphics,
            Plygon * polygon ) const  [private]
```

Draws a polygon shape using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
polygon	The Plygon object representing the polygon to be drawn.

Definition at line 270 of file Renderer.cpp.

```
270                                                                     {
```

```
271     ColorShape outline_color = polygon->getOutlineColor();
272     Gdiplus::Pen polygon_outline(
273         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
274                        outline_color.b),
275         polygon->getOutlineThickness());
276 
277     // Extract vertices and create an array of Gdiplus::PointF
278     Gdiplus::PointF* points = new Gdiplus::PointF[polygon->getPoints().size()];
279     int idx = 0;
280     const std::vector< Vector2Df >& vertices = polygon->getPoints();
281     for (const Vector2Df vertex : vertices) {
282         points[idx++] = Gdiplus::PointF(vertex.x, vertex.y);
283     }
284 
285     // Determine the fill mode based on the polygon's fill rule
286     Gdiplus::FillMode fill_mode;
287     if (polygon->getFillRule() == "evenodd") {
288         fill_mode = Gdiplus::FillModeAlternate;
289     } else if (polygon->getFillRule() == "nonzero") {
290         fill_mode = Gdiplus::FillModeWinding;
291     }
292 
293     // Create a bounding rectangle for the polygon
294     Vector2Df min_bound = polygon->getMinBound();
295     Vector2Df max_bound = polygon->getMaxBound();
296     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
297                          max_bound.y - min_bound.y);
298     // Get the fill brush for the polygon
299     Gdiplus::Brush* polygon_fill = getBrush(polygon, bound);
300 
301     // If the fill brush is a gradient, fill the polygon with a corner color
302     if (Gdiplus::PathGradientBrush* brush =
303             dynamic_cast< Gdiplus::PathGradientBrush* >(polygon_fill)) {
304         ColorShape color = polygon->getGradient()->getStops().back().getColor();
305         Gdiplus::SolidBrush corner_fill(
306             Gdiplus::Color(color.a, color.r, color.g, color.b));
307         graphics.FillPolygon(&corner_fill, points, idx, fill_mode);
308     }
309 
310     graphics.FillPolygon(polygon_fill, points, idx, fill_mode);
311     graphics.DrawPolygon(&polygon_outline, points, idx);
312 
313     delete[] points;
314     delete polygon_fill;
315 }
```

3.16.2.10 drawPolyline()

```
void Renderer::drawPolyline (
            Gdiplus::Graphics & graphics,
            Plyline * polyline ) const  [private]
```

Draws a polyline shape using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
polyline	The Plyline object representing the polyline to be drawn.

Definition at line 385 of file Renderer.cpp.

```
386                                                     {
387     ColorShape outline_color = polyline->getOutlineColor();
388     Gdiplus::Pen polyline_outline(
389         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
390                        outline_color.b),
391         polyline->getOutlineThickness());
392 
393     // Determine the fill mode based on the polyline's fill rule
394     Gdiplus::FillMode fill_mode;
395     if (polyline->getFillRule() == "evenodd") {
396         fill_mode = Gdiplus::FillModeAlternate;
```

```
397     } else if (polyline->getFillRule() == "nonzero") {
398         fill_mode = Gdiplus::FillModeWinding;
399     }
400
401     Gdiplus::GraphicsPath path(fill_mode);
402     const std::vector< Vector2Df >& points = polyline->getPoints();
403     if (points.size() < 2) {
404         return;
405     }
406
407     path.StartFigure();
408     path.AddLine(points[0].x, points[0].y, points[1].x, points[1].y);
409     for (size_t i = 2; i < points.size(); ++i) {
410         path.AddLine(points[i - 1].x, points[i - 1].y, points[i].x,
411                      points[i].y);
412     }
413
414     // Create a bounding rectangle for the polyline
415     Vector2Df min_bound = polyline->getMinBound();
416     Vector2Df max_bound = polyline->getMaxBound();
417     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
418                          max_bound.y - min_bound.y);
419     Gdiplus::Brush* polyline_fill = getBrush(polyline, bound);
420
421     // If the fill brush is a gradient, fill the polyline with a corner color
422     if (Gdiplus::PathGradientBrush* brush =
423             dynamic_cast< Gdiplus::PathGradientBrush* >(polyline_fill)) {
424         ColorShape color =
425             polyline->getGradient()->getStops().back().getColor();
426         Gdiplus::SolidBrush corner_fill(
427             Gdiplus::Color(color.a, color.r, color.g, color.b));
428         graphics.FillPath(&corner_fill, &path);
429     }
430
431     graphics.FillPath(polyline_fill, &path);
432     graphics.DrawPath(&polyline_outline, &path);
433
434     delete polyline_fill;
435 }
```

3.16.2.11 drawRectangle()

```
void Renderer::drawRectangle (
            Gdiplus::Graphics & graphics,
            Rect * rectangle ) const  [private]
```

Draws a rectangle shape using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
rectangle	The Rect object representing the rectangle to be drawn.

Definition at line 134 of file Renderer.cpp.

```
135                                                 {
136     float x = rectangle->getPosition().x;
137     float y = rectangle->getPosition().y;
138     float width = rectangle->getWidth();
139     float height = rectangle->getHeight();
140     ColorShape outline_color = rectangle->getOutlineColor();
141
142     // Create a pen for the rectangle outline
143     Gdiplus::Pen rect_outline(Gdiplus::Color(outline_color.a, outline_color.r,
144                                              outline_color.g, outline_color.b),
145                               rectangle->getOutlineThickness());
146     Gdiplus::RectF bound(x, y, width, height);
147     Gdiplus::Brush* rect_fill = getBrush(rectangle, bound);
148
149     // Check if the rectangle has rounded corners
150     if (rectangle->getRadius().x != 0 || rectangle->getRadius().y != 0) {
151         float dx = rectangle->getRadius().x * 2;
```

```
152         float dy = rectangle->getRadius().y * 2;
153
154         // Create a GraphicsPath for drawing rounded rectangles
155         Gdiplus::GraphicsPath path;
156         path.AddArc(x, y, dx, dy, 180, 90);
157         path.AddArc(x + width - dx, y, dx, dy, 270, 90);
158         path.AddArc(x + width - dx, y + height - dy, dx, dy, 0, 90);
159         path.AddArc(x, y + height - dy, dx, dy, 90, 90);
160         path.CloseFigure();
161         // Fill and draw the rounded rectangle
162         if (Gdiplus::PathGradientBrush* brush =
163                 dynamic_cast< Gdiplus::PathGradientBrush* >(rect_fill)) {
164             ColorShape color =
165                 rectangle->getGradient()->getStops().back().getColor();
166             Gdiplus::SolidBrush corner_fill(
167                 Gdiplus::Color(color.a, color.r, color.g, color.b));
168             graphics.FillPath(&corner_fill, &path);
169         }
170
171         graphics.FillPath(rect_fill, &path);
172         graphics.DrawPath(&rect_outline, &path);
173     } else {
174         // Fill and draw the regular rectangle
175         if (Gdiplus::PathGradientBrush* brush =
176                 dynamic_cast< Gdiplus::PathGradientBrush* >(rect_fill)) {
177             ColorShape color =
178                 rectangle->getGradient()->getStops().back().getColor();
179             Gdiplus::SolidBrush corner_fill(
180                 Gdiplus::Color(color.a, color.r, color.g, color.b));
181             graphics.FillRectangle(&corner_fill, x, y, width, height);
182         }
183
184         graphics.FillRectangle(rect_fill, x, y, width, height);
185         graphics.DrawRectangle(&rect_outline, x, y, width, height);
186     }
187
188     delete rect_fill;
189 }
```

3.16.2.12 drawText()

```
void Renderer::drawText (
            Gdiplus::Graphics & graphics,
            Text * text ) const  [private]
```

Draws text using Gdiplus::Graphics.

Parameters

graphics	The Gdiplus::Graphics context for drawing.
text	The Text object representing the text to be drawn.

Definition at line 318 of file Renderer.cpp.

```
318                                                                 {
319     ColorShape outline_color = text->getOutlineColor();
320     graphics.SetTextRenderingHint(Gdiplus::TextRenderingHintAntiAliasGridFit);
321
322     Gdiplus::Pen text_outline(Gdiplus::Color(outline_color.a, outline_color.r,
323                                              outline_color.g, outline_color.b),
324                               text->getOutlineThickness());
325
326     // Set the font family for the text
327     Gdiplus::FontFamily font_family(L"Times New Roman");
328
329     // Set the position for the text
330     Gdiplus::PointF position(text->getPosition().x, text->getPosition().y);
331     Gdiplus::GraphicsPath path;
332
333     // Convert the content to wide string for GDI+
334     std::wstring_convert< std::codecvt_utf8_utf16< wchar_t > > converter;
335     std::wstring wide_content = converter.from_bytes(text->getContent());
```

```
336
337     // Set text alignment based on anchor position
338     Gdiplus::StringFormat string_format;
339     if (text->getAnchor() == "middle") {
340         string_format.SetAlignment(Gdiplus::StringAlignmentCenter);
341         position.X += 7;
342     } else if (text->getAnchor() == "end") {
343         string_format.SetAlignment(Gdiplus::StringAlignmentFar);
344         position.X += 14;
345     } else {
346         string_format.SetAlignment(Gdiplus::StringAlignmentNear);
347     }
348
349     // Set font style based on text style
350     Gdiplus::FontStyle font_style = Gdiplus::FontStyleRegular;
351     if (text->getFontStyle() == "italic" || text->getFontStyle() == "oblique") {
352         font_style = Gdiplus::FontStyleItalic;
353         position.Y -= 1;
354     }
355
356     path.AddString(wide_content.c_str(), wide_content.size(), &font_family,
357                    font_style, text->getFontSize(), position, &string_format);
358     Gdiplus::RectF bound;
359     path.GetBounds(&bound);
360     Gdiplus::Brush* text_fill = getBrush(text, bound);
361
362     // If the fill brush is a gradient, fill the text with a corner color
363     if (Gdiplus::PathGradientBrush* brush =
364             dynamic_cast< Gdiplus::PathGradientBrush* >(text_fill)) {
365         ColorShape color = text->getGradient()->getStops().back().getColor();
366         Gdiplus::SolidBrush corner_fill(
367             Gdiplus::Color(color.a, color.r, color.g, color.b));
368         graphics.FillPath(&corner_fill, &path);
369     }
370
371     graphics.FillPath(text_fill, &path);
372     if (text->getOutlineColor().a != 0 &&
373         text->getOutlineColor().a == text->getFillColor().a) {
374         text_outline.SetColor(Gdiplus::Color(255, 255, 255, 255));
375         graphics.DrawPath(&text_outline, &path);
376         text_outline.SetColor(Gdiplus::Color(outline_color.a, outline_color.r,
377                                              outline_color.g, outline_color.b));
378     }
379     graphics.DrawPath(&text_outline, &path);
380
381     delete text_fill;
382 }
```

3.16.2.13 getBrush()

```
Gdiplus::Brush * Renderer::getBrush (
            SVGElement * shape,
            Gdiplus::RectF bound ) const  [private]
```

Gets the Gdiplus::brush object for the shape fill.

Parameters

shape	The SVGElement representing the shape.
bound	The bounding box of the shape.

Returns

The Gdiplus::brush object for the shape fill.

Definition at line 683 of file Renderer.cpp.

```
684                                                     {
685     Gradient* gradient = shape->getGradient();
```

```
686     if (gradient != NULL) {
687         std::pair< Vector2Df, Vector2Df > points = gradient->getPoints();
688         std::vector< Stop > stops = gradient->getStops();
689         int stop_size = stops.size() + 2;
690         Gdiplus::Color* colors = new Gdiplus::Color[stop_size];
691         float* offsets = new float[stop_size];
692 
693         if (gradient->getClass() == "LinearGradient") {
694             // Brush linear gradient
695             if (gradient->getUnits() == "objectBoundingBox") {
696                 points.first.x = bound.X;
697                 points.first.y = bound.Y;
698                 points.second.x = bound.X + bound.Width;
699                 points.second.y = bound.Y + bound.Height;
700             }
701 
702             // Set the center color
703             offsets[0] = 0;
704             offsets[stop_size - 1] = 1;
705             colors[0] =
706                 Gdiplus::Color(stops[0].getColor().a, stops[0].getColor().r,
707                                stops[0].getColor().g, stops[0].getColor().b);
708             colors[stop_size - 1] =
709                 Gdiplus::Color(stops[stop_size - 3].getColor().a,
710                                stops[stop_size - 3].getColor().r,
711                                stops[stop_size - 3].getColor().g,
712                                stops[stop_size - 3].getColor().b);
713 
714             // Reverse the order of the stops
715             for (size_t i = 1; i < stop_size - 1; ++i) {
716                 colors[i] = Gdiplus::Color(
717                     stops[i - 1].getColor().a, stops[i - 1].getColor().r,
718                     stops[i - 1].getColor().g, stops[i - 1].getColor().b);
719                 offsets[i] = stops[i - 1].getOffset();
720             }
721 
722             // Create the brush of linear gradient
723             Gdiplus::LinearGradientBrush* fill =
724                 new Gdiplus::LinearGradientBrush(
725                     Gdiplus::PointF(points.first.x, points.first.y),
726                     Gdiplus::PointF(points.second.x, points.second.y),
727                     colors[0], colors[stop_size - 1]);
728             fill->SetWrapMode(Gdiplus::WrapModeTileFlipX);
729             fill->SetInterpolationColors(colors, offsets, stop_size);
730             applyTransformsOnBrush(gradient->getTransforms(), fill);
731 
732             delete[] colors;
733             delete[] offsets;
734             return fill;
735         } else if (gradient->getClass() == "RadialGradient") {
736             // Brush radiol gradient
737             RadialGradient* radial_gradient =
738                 dynamic_cast< RadialGradient* >(gradient);
739             Vector2Df radius = radial_gradient->getRadius();
740 
741             // If the gradient is in userSpaceOnUse, the radius is the distance
742             if (gradient->getUnits() == "userSpaceOnUse") {
743                 bound.X = points.first.x - radius.x;
744                 bound.Y = points.first.y - radius.x;
745                 bound.Width = radius.x * 2;
746                 bound.Height = radius.x * 2;
747             }
748 
749             Gdiplus::GraphicsPath path;
750             path.AddEllipse(bound);
751             Gdiplus::PathGradientBrush* fill =
752                 new Gdiplus::PathGradientBrush(&path);
753 
754             // Set the center color
755             offsets[0] = 0;
756             offsets[stop_size - 1] = 1;
757             colors[0] = Gdiplus::Color(stops[stop_size - 3].getColor().a,
758                                        stops[stop_size - 3].getColor().r,
759                                        stops[stop_size - 3].getColor().g,
760                                        stops[stop_size - 3].getColor().b);
761             colors[stop_size - 1] =
762                 Gdiplus::Color(stops[0].getColor().a, stops[0].getColor().r,
763                                stops[0].getColor().g, stops[0].getColor().b);
764 
765             // Reverse the order of the stops
766             for (size_t i = 1; i < stop_size - 1; ++i) {
767                 colors[i] =
768                     Gdiplus::Color(stops[stop_size - 2 - i].getColor().a,
769                                    stops[stop_size - 2 - i].getColor().r,
770                                    stops[stop_size - 2 - i].getColor().g,
771                                    stops[stop_size - 2 - i].getColor().b);
772                 offsets[i] = 1 - stops[stop_size - 2 - i].getOffset();
```

```
773             }
774
775             fill->SetInterpolationColors(colors, offsets, stop_size);
776             applyTransformsOnBrush(gradient->getTransforms(), fill);
777             delete[] colors;
778             delete[] offsets;
779             return fill;
780         }
781     } else {
782         ColorShape color = shape->getFillColor();
783         Gdiplus::SolidBrush* fill = new Gdiplus::SolidBrush(
784             Gdiplus::Color(color.a, color.r, color.g, color.b));
785         return fill;
786     }
787     return nullptr;
788 }
```

3.16.2.14 getInstance()

```
Renderer * Renderer::getInstance ( )  [static]
```

Gets the singleton instance of the Renderer class.

Returns

The singleton instance of the Renderer class.

Definition at line 11 of file Renderer.cpp.

```
11                                  {
12     if (instance == nullptr) {
13         instance = new Renderer();
14     }
15     return instance;
16 }
```

The documentation for this class was generated from the following files:

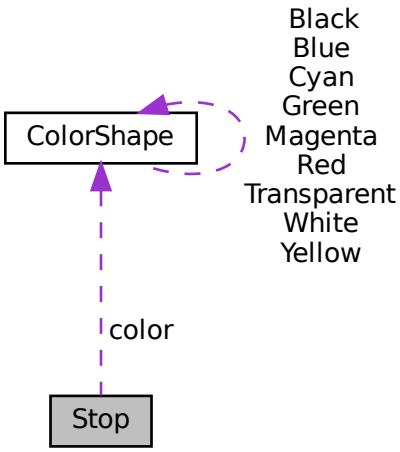
- src/Renderer.hpp
- src/Renderer.cpp

3.17 Stop Class Reference

A class that represents a stop.

```
#include <Stop.hpp>
```

Collaboration diagram for Stop:



Public Member Functions

- Stop (const ColorShape &color, float offset)

  Constructs a Stop object.
- ColorShape getColor () const

  Gets the color of the stop.
- float getOffset () const

  Gets the offset of the stop.

Private Attributes

- ColorShape color

  The color of the stop.
- float offset

  The offset of the stop.

3.17.1 Detailed Description

A class that represents a stop.

The Stop class represents a stop. It contains a color and an offset.

Definition at line 11 of file Stop.hpp.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 Stop()

```
Stop::Stop (
            const ColorShape & color,
            float offset )
```

Constructs a Stop object.

Parameters

color	The color of the stop.
offset	The offset of the stop.

Definition at line 3 of file Stop.cpp.

```
4     : color(color), offset(offset) {}
```

3.17.3 Member Function Documentation

3.17.3.1 getColor()

```
ColorShape Stop::getColor ( ) const
```

Gets the color of the stop.

Returns

The color of the stop.

Definition at line 6 of file Stop.cpp.

```
6 { return color; }
```

3.17.3.2 getOffset()

```
float Stop::getOffset ( ) const
```

Gets the offset of the stop.

Returns

The offset of the stop.

Definition at line 8 of file Stop.cpp.

```
8 { return offset; }
```

The documentation for this class was generated from the following files:

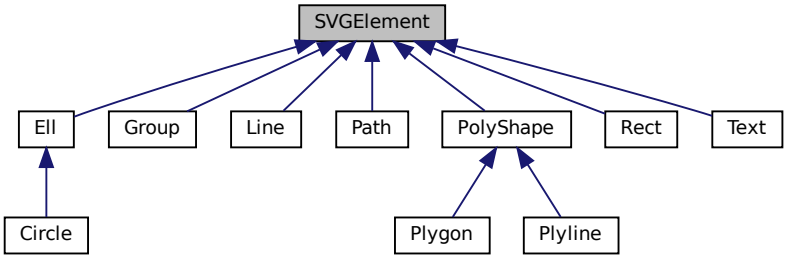
- src/graphics/Stop.hpp
- src/graphics/Stop.cpp

3.18 SVGElement Class Reference

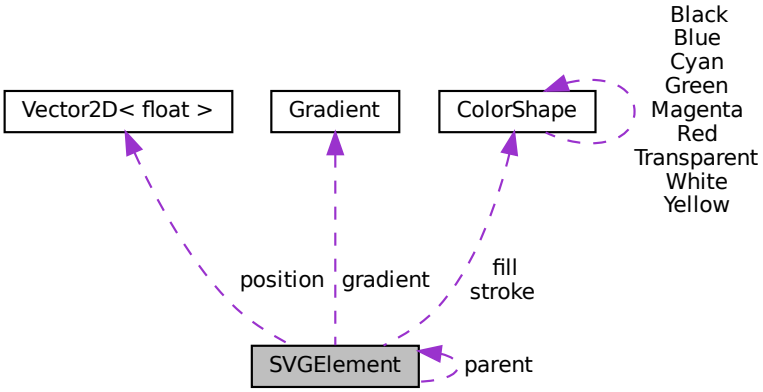
Represents an element in an SVG file.

`#include <SVGElement.hpp>`

Inheritance diagram for SVGElement:



Collaboration diagram for SVGElement:



Public Member Functions

- virtual ~SVGElement ()=default

  Virtual constructor.
- virtual std::string getClass () const =0

  Gets the type of the shape.
- void setFillColor (const ColorShape &color)

  Sets the fill color of the shape.
- void setOutlineColor (const ColorShape &color)

  Sets the outline color of the shape.

- void setOutlineThickness (float thickness)

  Sets the outline thickness of the shape.
- void setPosition (float x, float y)

  Sets the position of the shape.
- void setPosition (const Vector2Df &position)

  Sets the position of the shape.
- const ColorShape & getFillColor () const

  Gets the fill color of the shape.
- const ColorShape & getOutlineColor () const

  Gets the outline color of the shape.
- float getOutlineThickness () const

  Gets the outline thickness of the shape.
- Vector2Df getPosition () const

  Get the current position of the shape.
- virtual Vector2Df getMinBound () const

  Gets the minimum bound of the shape.
- virtual Vector2Df getMaxBound () const

  Gets the maximum bound of the shape.
- virtual void printData () const

  Prints the data of the shape.
- void setTransforms (const std::vector< std::string > &transforms)

  Sets the transformations of the shape.
- std::vector< std::string > getTransforms () const

  Gets the transformations of the shape.
- void setParent (SVGElement *parent)

  Parent pointer setter to make the composite design pattern.
- SVGElement * getParent () const

  Parent pointer getter.
- void setGradient (Gradient *gradient)

  Sets the gradient of the shape.
- Gradient * getGradient () const

  Gets the gradient of the shape.
- virtual void addElement (SVGElement *element)

  Adds a shape to the composite group.

Protected Member Functions

- SVGElement ()

  Constructs a Shape object.
- SVGElement (const ColorShape &fill, const ColorShape &stroke, float stroke_width)

  Constructs a Shape object.
- SVGElement (const ColorShape &fill, const ColorShape &stroke, float stroke_width, const Vector2Df &position)

  Constructs a Shape object.

Protected Attributes

- SVGElement * parent

  Pointer to the group that contains the shape.

Private Attributes

- ColorShape fill
  Fill color.
- ColorShape stroke
  Outline color.
- float stroke_width
  Thickness of the shape's outline.
- Vector2Df position
  Position of the shape.
- std::vector< std::string > transforms
  List of transformations.
- Gradient * gradient
  Pointer to the gradient that contains the shape.

3.18.1 Detailed Description

Represents an element in an SVG file.

Note

This class is abstract and cannot be instantiated.

This class is applied Abstract Factory design pattern and used as interface for other shapes.

This class is applied Composite design pattern and used as base class for other shapes.

Definition at line 18 of file SVGElement.hpp.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 SVGElement() [1/3]

```
SVGElement::SVGElement ( )  [protected]
```

Constructs a Shape object.

Note

This constructor is protected because Shape is an abstract class that cannot be instantiated.

Definition at line 5 of file SVGElement.cpp.

```
6     : fill(ColorShape::Black), stroke(ColorShape::Transparent), stroke_width(1),
7       gradient(NULL) {}
```

3.18.2.2 SVGElement() [2/3]

```
SVGElement::SVGElement (
            const ColorShape & fill,
            const ColorShape & stroke,
            float stroke_width )  [protected]
```

Constructs a Shape object.

Parameters

fill	The fill color of the shape
stroke	The outline color of the shape
stroke_width	The outline thickness of the shape

Note

This constructor is protected because Shape is an abstract class that cannot be instantiated.

Definition at line 9 of file SVGElement.cpp.

```
11     : fill(fill), stroke(stroke), stroke_width(stroke_width), gradient(NULL) {}
```

3.18.2.3 SVGElement() [3/3]

```
SVGElement::SVGElement (
            const ColorShape & fill,
            const ColorShape & stroke,
            float stroke_width,
            const Vector2Df & position )  [protected]
```

Constructs a Shape object.

Parameters

fill	The fill color of the shape
stroke	The outline color of the shape
stroke_width	The outline thickness of the shape
position	The position of the shape

Note

This constructor is protected because Shape is an abstract class that cannot be instantiated.

Definition at line 13 of file SVGElement.cpp.

```
15     : fill(fill), stroke(stroke), stroke_width(stroke_width),
16       position(position), gradient(NULL) {}
```

3.18.3 Member Function Documentation

3.18.3.1 addElement()

```
void SVGElement::addElement (
            SVGElement * element )  [virtual]
```

Adds a shape to the composite group.

Parameters

element	The shape to be added to the composite group.

Note

This function is used for composite design pattern

This function is virtual and can be overridden by derived classes.

Reimplemented in Group.

Definition at line 83 of file SVGElement.cpp.

```
83 {}
```

3.18.3.2 getClass()

```
virtual std::string SVGElement::getClass ( ) const  [pure virtual]
```

Gets the type of the shape.

Returns

The type of the shape

Note

This function is used for determining the type of the shape.

This function is pure virtual and must be implemented by derived classes.

Implemented in Text, Rect, Plyline, Plygon, Path, Line, Group, Ell, Circle, and PolyShape.

3.18.3.3 getFillColor()

```
const ColorShape & SVGElement::getFillColor ( ) const
```

Gets the fill color of the shape.

Returns

The fill color of the shape.

Note

The default fill color is white.

Definition at line 20 of file SVGElement.cpp.

```
20 { return fill; }
```

3.18.3.4 getGradient()

```
Gradient * SVGElement::getGradient ( ) const
```

Gets the gradient of the shape.

Returns

The gradient of the shape.

Note

The default gradient of the shape is NULL.

Definition at line 81 of file SVGElement.cpp.

```
81 { return gradient; }
```

3.18.3.5 getMaxBound()

```
Vector2Df SVGElement::getMaxBound ( ) const  [virtual]
```

Gets the maximum bound of the shape.

Returns

The maximum bound of the shape.

Reimplemented in PolyShape, and Ell.

Definition at line 45 of file SVGElement.cpp.

```
45 { return Vector2Df(); }
```

3.18.3.6 getMinBound()

```
Vector2Df SVGElement::getMinBound ( ) const  [virtual]
```

Gets the minimum bound of the shape.

Returns

The minimum bound of the shape.

Reimplemented in PolyShape, and Ell.

Definition at line 43 of file SVGElement.cpp.

```
43 { return Vector2Df(); }
```

3.18.3.7 getOutlineColor()

```
const ColorShape & SVGElement::getOutlineColor ( ) const
```

Gets the outline color of the shape.

Returns

The outline color of the shape.

Note

The default outline color is white.

Definition at line 24 of file SVGElement.cpp.

```
24 { return stroke; }
```

3.18.3.8 getOutlineThickness()

```
float SVGElement::getOutlineThickness ( ) const
```

Gets the outline thickness of the shape.

Returns

The outline thickness of the shape.

Note

The default outline thickness is 0.

Definition at line 30 of file SVGElement.cpp.

```
30 { return stroke_width; }
```

3.18.3.9 getParent()

```
SVGElement * SVGElement::getParent ( ) const
```

Parent pointer getter.

Returns

The parent pointer

Note

This function is used for composite design pattern

Definition at line 77 of file SVGElement.cpp.

```
77 { return parent; }
```

3.18.3.10 getPosition()

```
Vector2Df SVGElement::getPosition ( ) const
```

Get the current position of the shape.

Returns

The current position of the shape

Note

The default position of the shape is (0, 0).

Definition at line 41 of file SVGElement.cpp.

```
41 { return position; }
```

3.18.3.11 getTransforms()

```
std::vector< std::string > SVGElement::getTransforms ( ) const
```

Gets the transformations of the shape.

Returns

The transformations of the shape.

Note

The default transformations of the shape is empty.

The transformations can be either "translate", "rotate", "scale",

Definition at line 71 of file SVGElement.cpp.

```
71                                                  {
72     return transforms;
73 }
```

3.18.3.12 printData()

```
void SVGElement::printData ( ) const  [virtual]
```

Prints the data of the shape.

Note

This function is used for debugging purposes.

This function is virtual and can be overridden by derived classes.

Reimplemented in Text, Rect, PolyShape, Path, Group, and Ell.

Definition at line 47 of file SVGElement.cpp.

```
47                                  {
48     std::cout « "Shape: " « getClass() « std::endl;
49     std::cout « "Fill: " « getFillColor() « std::endl;
50     std::cout « "Stroke: " « getOutlineColor() « std::endl;
51     std::cout « "Stroke width: " « getOutlineThickness() « std::endl;
52     std::cout « "Position: " « getPosition().x « " " « getPosition().y
53               « std::endl;
54     std::cout « "Transforms: ";
55     for (auto transform : transforms) {
56         std::cout « transform « " ";
57     }
58     std::cout « std::endl;
59     if (gradient != NULL)
60         std::cout « "Gradient: " « gradient->getClass() « " "
61                   « gradient->getPoints().first.x « " "
62                   « gradient->getPoints().first.y « " "
63                   « gradient->getPoints().second.x « " "
64                   « gradient->getPoints().second.y « std::endl;
65 }
```

3.18.3.13 setFillColor()

```
void SVGElement::setFillColor (
             const ColorShape & color )
```

Sets the fill color of the shape.

Parameters

color	The new fill color of the shape.

Definition at line 18 of file SVGElement.cpp.

```
18 { fill = color; }
```

3.18.3.14 setGradient()

```
void SVGElement::setGradient (
             Gradient * gradient )
```

Sets the gradient of the shape.

Parameters

gradient	The new gradient of the shape.

Note

The default gradient of the shape is NULL.

Definition at line 79 of file SVGElement.cpp.

```
79 { this->gradient = gradient; }
```

3.18.3.15 setOutlineColor()

```
void SVGElement::setOutlineColor (
            const ColorShape & color )
```

Sets the outline color of the shape.

Parameters

color	The new outline color of the shape.

Definition at line 22 of file SVGElement.cpp.

```
22 { stroke = color; }
```

3.18.3.16 setOutlineThickness()

```
void SVGElement::setOutlineThickness (
            float thickness )
```

Sets the outline thickness of the shape.

Parameters

thickness	The new outline thickness of the shape.

Note

If the thickness is negative, the outline will be inside the shape. If the thickness is positive, the outline will be outside the shape. If the thickness is zero, no outline will be drawn.

The default outline thickness is 0.

The outline thickness cannot be greater than the radius of the shape.

Definition at line 26 of file SVGElement.cpp.

```
26                                                  {
```

```
27     stroke_width = thickness;
28 }
```

3.18.3.17 setParent()

```
void SVGElement::setParent (
            SVGElement * parent )
```

Parent pointer setter to make the composite design pattern.

Parameters

parent	The parent pointer

Note

This function is used for composite design pattern

Definition at line 75 of file SVGElement.cpp.

```
75 { this->parent = parent; }
```

3.18.3.18 setPosition() [1/2]

```
void SVGElement::setPosition (
            const Vector2Df & position )
```

Sets the position of the shape.

Parameters

position	The new position of the shape (Vector2f is a typedef of coordination vector)

Note

The default position of the shape is (0, 0).

The position of the shape is relative to its origin.

Definition at line 37 of file SVGElement.cpp.

```
37                                                  {
38     setPosition(position.x, position.y);
39 }
```

3.18.3.19 setPosition() [2/2]

```
void SVGElement::setPosition (
            float x,
            float y )
```

Sets the position of the shape.

Parameters

x	The x coordinate of the new position
y	The y coordinate of the new position

Note

The default position of the shape is (0, 0).

The position of the shape is relative to its origin.

Definition at line 32 of file SVGElement.cpp.

```
32                                          {
33     position.x = x;
34     position.y = y;
35 }
```

3.18.3.20 setTransforms()

```
void SVGElement::setTransforms (
            const std::vector< std::string > & transforms )
```

Sets the transformations of the shape.

Parameters

transforms	The new transformations of the shape.

Note

The default transformations of the shape is empty.

The transformations can be either "translate", "rotate", "scale",

Definition at line 67 of file SVGElement.cpp.

```
67                                                                  {
68     this->transforms = transforms;
69 }
```

The documentation for this class was generated from the following files:

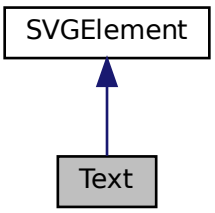
- src/graphics/SVGElement.hpp
- src/graphics/SVGElement.cpp

3.19 Text Class Reference

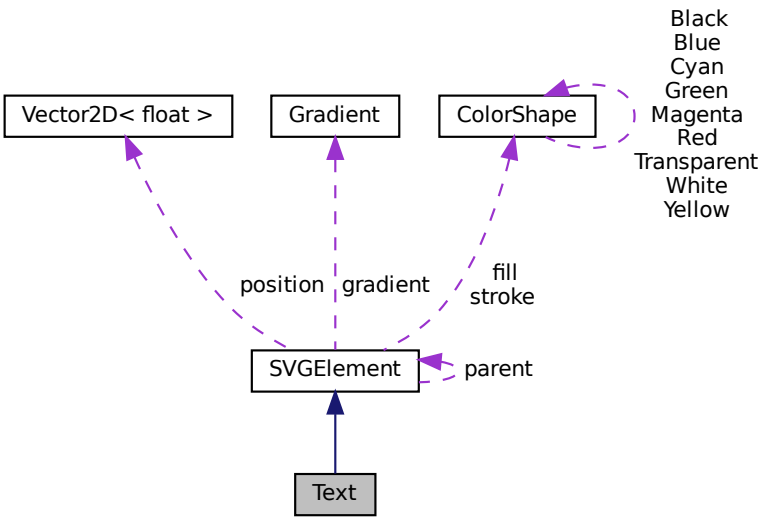
Represents text in 2D space.

`#include <Text.hpp>`

Inheritance diagram for Text:



Collaboration diagram for Text:



Public Member Functions

- Text (Vector2Df pos, std::string text, float font_size, const ColorShape &fill, const ColorShape &stroke, float stroke_width)

  Constructs a Text object.
- std::string getClass () const override

Gets the type of the shape.

- void setContent (std::string content)

  Sets the string of the text.
- std::string getContent () const

  Gets the string of the text.
- void setFontSize (float font_size)

  Sets the font size of the text.
- float getFontSize () const

  Gets the font size of the text.
- void setAnchor (std::string anchor)

  Sets the anchor of the text.
- std::string getAnchor () const

  Gets the anchor of the text.
- void setFontStyle (std::string style)

  Sets the style of the text.
- std::string getFontStyle () const

  Gets the style of the text.
- void printData () const override

  Prints the data of the text.

Private Attributes

- std::string content

  Text element.
- float font_size

  Font size of the text.
- std::string anchor

  Anchor of the text.
- std::string style

  Style of the text.

Additional Inherited Members

3.19.1 Detailed Description

Represents text in 2D space.

The Text class is derived from the SVGElement class and defines a text element with a specified position, string, fill color, and font size.

Definition at line 12 of file Text.hpp.

3.19.2 Constructor & Destructor Documentation

3.19.2.1 Text()

```
Text::Text (
            Vector2Df pos,
            std::string text,
            float font_size,
            const ColorShape & fill,
            const ColorShape & stroke,
            float stroke_width )
```

Constructs a Text object.

Parameters

pos	The position of the text.
text	The string of the text.
fill	The fill color of the text
font_size	The font size of the text (default is 1).

Definition at line 3 of file Text.cpp.

```
5     : SVGElement(fill, stroke, stroke_width, pos), content(text),
6       font_size(font_size) {}
```

3.19.3 Member Function Documentation

3.19.3.1 getAnchor()

```
std::string Text::getAnchor ( ) const
```

Gets the anchor of the text.

Returns

The anchor of the text.

Definition at line 20 of file Text.cpp.

```
20 { return anchor; }
```

3.19.3.2 getClass()

```
std::string Text::getClass ( ) const  [override], [virtual]
```

Gets the type of the shape.

Returns

The string "Text".

Implements SVGElement.

Definition at line 8 of file Text.cpp.

```
8 { return "Text"; }
```

3.19.3.3 getContent()

```
std::string Text::getContent ( ) const
```

Gets the string of the text.

Returns

The string of the text.

Definition at line 16 of file Text.cpp.

```
16 { return content; }
```

3.19.3.4 getFontSize()

```
float Text::getFontSize ( ) const
```

Gets the font size of the text.

Returns

The font size of the text.

Definition at line 12 of file Text.cpp.

```
12 { return font_size; }
```

3.19.3.5 getFontStyle()

```
std::string Text::getFontStyle ( ) const
```

Gets the style of the text.

Returns

The style of the text.

Definition at line 24 of file Text.cpp.

```
24 { return style; }
```

3.19.3.6 setAnchor()

```
void Text::setAnchor (
            std::string anchor )
```

Sets the anchor of the text.

Parameters

anchor	The new anchor of the text.

Definition at line 18 of file Text.cpp.

```
18 { this->anchor = anchor; }
```

3.19.3.7 setContent()

```
void Text::setContent (
            std::string content )
```

Sets the string of the text.

Parameters

content	The new string of the text.

Definition at line 14 of file Text.cpp.

```
14 { this->content = content; }
```

3.19.3.8 setFontSize()

```
void Text::setFontSize (
            float font_size )
```

Sets the font size of the text.

Parameters

font_size	The new font size of the text.

Definition at line 10 of file Text.cpp.

```
10 { this->font_size = font_size; }
```

3.19.3.9 setFontStyle()

```
void Text::setFontStyle (
            std::string style )
```

Sets the style of the text.

Parameters

style	The new style of the text.

Definition at line 22 of file Text.cpp.

```
22 { this->style = font_style; }
```

The documentation for this class was generated from the following files:

- src/graphics/Text.hpp
- src/graphics/Text.cpp

3.20 Vector2D< T > Class Template Reference

Utility template class for manipulating 2-dimensional vectors.

`#include <Vector2D.hpp>`

Public Member Functions

- Vector2D ()

  Default constructor.
- Vector2D (T X, T Y)

  Construct the vector from its coordinates.
- template<typename U >
  Vector2D (const Vector2D< U > &vector)

  Construct the vector from another type of vector.

Public Attributes

- T x

  X coordinate of the vector.
- T y

  Y coordinate of the vector.

3.20.1 Detailed Description

template<typename T>
class Vector2D< T >

Utility template class for manipulating 2-dimensional vectors.

Vector2D is a simple class that defines a mathematical vector with two coordinates (x and y). It can be used to represent anything that has two dimensions: a size, a point, a velocity, etc.

The template parameter T is the type of the coordinates. It can be any type that supports arithmetic operations (+, -, /, *) and comparisons (==, !=), for example int or float.

Definition at line 17 of file Vector2D.hpp.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 Vector2D() [1/3]

```
template<typename T >
Vector2D< T >::Vector2D  [inline]
```

Default constructor.

Creates a Vector2(0, 0).

Definition at line 197 of file Vector2D.hpp.

```
197 : x(0), y(0) {}
```

3.20.2.2 Vector2D() [2/3]

```
template<typename T >
Vector2D< T >::Vector2D (
            T X,
            T Y )  [inline]
```

Construct the vector from its coordinates.

Parameters

X	X coordinate
Y	Y coordinate

Definition at line 200 of file Vector2D.hpp.

```
200 : x(X), y(Y) {}
```

3.20.2.3 Vector2D() [3/3]

```
template<typename T >
template<typename U >
Vector2D< T >::Vector2D (
            const Vector2D< U > & vector )  [inline], [explicit]
```

Construct the vector from another type of vector.

This constructor doesn't replace the copy constructor, it's called only when U != T. A call to this constructor will fail to compile if U is not convertible to T.

Definition at line 204 of file Vector2D.hpp.

```
205     : x(static_cast< T >(vector.x)), y(static_cast< T >(vector.y)) {}
```

The documentation for this class was generated from the following file:

- src/graphics/Vector2D.hpp

3.21 ViewBox Class Reference

A ViewBox is a rectangle that defines the area of the SVG canvas that should be visible to the user.

`#include <ViewBox.hpp>`

Public Member Functions

- ViewBox ()
  Default constructor.
- ViewBox (float X, float Y, float W, float H)
  Construct the ViewBox from its coordinates.
- float getX () const
  Get the X coordinate of the ViewBox.
- float getY () const
  Get the Y coordinate of the ViewBox.
- float getWidth () const
  Get the width of the ViewBox.
- float getHeight () const
  Get the height of the ViewBox.

Private Attributes

- float x
  X coordinate of the ViewBox.
- float y
  Y coordinate of the ViewBox.
- float w
  Width of the ViewBox.
- float h
  Height of the ViewBox.

3.21.1 Detailed Description

A ViewBox is a rectangle that defines the area of the SVG canvas that should be visible to the user.

The ViewBox is defined by its X and Y coordinates, its width and its height.

Definition at line 10 of file ViewBox.hpp.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 ViewBox() [1/2]

```
ViewBox::ViewBox ( )
```

Default constructor.

Creates a ViewBox(0, 0, 0, 0).

Definition at line 3 of file ViewBox.cpp.

```
3 : x(0), y(0), w(0), h(0) {}
```

3.21.2.2 ViewBox() [2/2]

```
ViewBox::ViewBox (
            float X,
            float Y,
            float W,
            float H )
```

Construct the ViewBox from its coordinates.

Parameters

X	X coordinate
Y	Y coordinate
W	Width
H	Height

Definition at line 5 of file ViewBox.cpp.

```
5 : x(X), y(Y), w(W), h(H) {}
```

3.21.3 Member Function Documentation

3.21.3.1 getHeight()

```
float ViewBox::getHeight ( ) const
```

Get the height of the ViewBox.

Returns

Height of the ViewBox

Definition at line 13 of file ViewBox.cpp.

```
13 { return h; }
```

3.21.3.2 getWidth()

```
float ViewBox::getWidth ( ) const
```

Get the width of the ViewBox.

Returns

Width of the ViewBox

Definition at line 11 of file ViewBox.cpp.

```
11 { return w; }
```

3.21.3.3 getX()

```
float ViewBox::getX ( ) const
```

Get the X coordinate of the ViewBox.

Returns

X coordinate of the ViewBox

Definition at line 7 of file ViewBox.cpp.

```
7 { return x; }
```

3.21.3.4 getY()

```
float ViewBox::getY ( ) const
```

Get the Y coordinate of the ViewBox.

Returns

Y coordinate of the ViewBox

Definition at line 9 of file ViewBox.cpp.

```
9 { return y; }
```

The documentation for this class was generated from the following files:

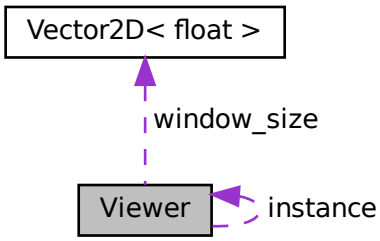
- src/graphics/ViewBox.hpp
- src/graphics/ViewBox.cpp

3.22 Viewer Class Reference

Represents a viewer for rendering and interacting with a scene.

`#include <Viewer.hpp>`

Collaboration diagram for Viewer:



Public Member Functions

- ~Viewer ()
  Destructor for the Viewer class.
- void handleMouseEvent (UINT message, WPARAM wParam, LPARAM lParam)
  Handles mouse events, such as wheel, move, left button down, and left button up.
- void handleKeyEvent (WPARAM wParam)
  Handles keyboard events.
- void getWindowSize (HWND hWnd) const
  Get the current window size.

Static Public Member Functions

- static Viewer * getInstance ()
  Gets the singleton instance of the Viewer class.

Public Attributes

- float offset_x
  X-coordinate offset of the viewer.
- float offset_y
  Y-coordinate offset of the viewer.
- float zoom_factor
  Zoom factor for scaling the view.
- float rotate_angle
  Rotation angle of the view.
- bool needs_repaint
- Vector2Df window_size
  Size of the window.

Private Member Functions

- Viewer ()

  Private constructor for the Viewer class.
- Viewer (const Viewer &)=delete

  Copy constructor for the Viewer class (deleted to enforce singleton pattern).
- void operator= (const Viewer &)=delete

  Copy assignment operator for the Viewer class (deleted to enforce singleton pattern).
- void handleMouseWheel (WPARAM wParam)

  Handles the mouse wheel event for zooming.
- void handleMouseMove (LPARAM lParam)

  Handles the mouse move event for panning.
- void handleLeftButtonDown (LPARAM lParam)

  Handles the left button down event for initiating dragging.
- void handleLeftButtonUp ()

  Handles the left button up event for ending dragging.
- void handleKeyDown (WPARAM wParam)

  Handles the key down event for rotating.

Private Attributes

- bool is_dragging

  Flag indicating whether the mouse is being dragged.
- POINT last_mouse_pos

  Last recorded mouse position.

Static Private Attributes

- static Viewer * instance = nullptr

  Singleton instance of the Viewer class.

3.22.1 Detailed Description

Represents a viewer for rendering and interacting with a scene.

The viewer supports the following interactions:

- Rotation: Press 'Q' to rotate the view counterclockwise and 'E' to rotate clockwise.
- Zooming: Use the scroll wheel to zoom in and out of the scene.
- Translation: Click and drag the left mouse button to translate the view.

Definition at line 16 of file Viewer.hpp.

3.22.2 Member Function Documentation

3.22.2.1 getInstance()

```
Viewer * Viewer::getInstance ( )  [static]
```

Gets the singleton instance of the Viewer class.

Returns

The singleton instance of the Viewer class.

Definition at line 4 of file Viewer.cpp.

```
4                                {
5     if (!instance) {
6         instance = new Viewer();
7     }
8     return instance;
9 }
```

3.22.2.2 getWindowSize()

```
void Viewer::getWindowSize (
            HWND hWnd ) const
```

Get the current window size.

Parameters

hWnd	The handle to the window.

Definition at line 103 of file Viewer.cpp.

```
103                                          {
104     RECT rect;
105     GetClientRect(hWnd, &rect);
106     instance->window_size.x = static_cast< float >(rect.right - rect.left);
107     instance->window_size.y = static_cast< float >(rect.bottom - rect.top);
108 }
```

3.22.2.3 handleKeyDown()

```
void Viewer::handleKeyDown (
            WPARAM wParam )  [private]
```

Handles the key down event for rotating.

Parameters

wParam	The WPARAM parameter of the message.

Definition at line 90 of file Viewer.cpp.

```
90                                      {
```

```
91      char key = static_cast< char >(wParam);
92      switch (tolower(key)) {
93          case 'q':
94              rotate_angle -= 1.0f;
95              break;
96
97          case 'e':
98              rotate_angle += 1.0f;
99              break;
100     }
101 }
```

3.22.2.4 handleKeyEvent()

```
void Viewer::handleKeyEvent (
            WPARAM wParam )
```

Handles keyboard events.

Parameters

wParam	The WPARAM parameter of the message.

Definition at line 47 of file Viewer.cpp.

```
47 { handleKeyDown(wParam); }
```

3.22.2.5 handleLeftButtonDown()

```
void Viewer::handleLeftButtonDown (
            LPARAM lParam )  [private]
```

Handles the left button down event for initiating dragging.

Parameters

lParam	The LPARAM parameter of the message.

Definition at line 74 of file Viewer.cpp.

```
74                                          {
75      is_dragging = true;
76      last_mouse_pos.x = static_cast< int >(LOWORD(lParam));
77      last_mouse_pos.y = static_cast< int >(HIWORD(lParam));
78      SetCapture(GetActiveWindow());
79  }
```

3.22.2.6 handleMouseEvent()

```
void Viewer::handleMouseEvent (
            UINT message,
```

```
            WPARAM wParam,
            LPARAM lParam )
```

Handles mouse events, such as wheel, move, left button down, and left button up.

Parameters

message	The Windows message identifier.
wParam	The WPARAM parameter of the message.
lParam	The LPARAM parameter of the message.

Definition at line 26 of file Viewer.cpp.

```
26                                                                      {
27     switch (message) {
28         case WM_MOUSEWHEEL:
29             handleMouseWheel(wParam);
30             break;
31
32         case WM_MOUSEMOVE:
33             if (wParam & MK_LBUTTON) {
34                 handleMouseMove(lParam);
35             }
36
37         case WM_LBUTTONDOWN:
38             handleLeftButtonDown(lParam);
39             break;
40
41         case WM_LBUTTONUP:
42             handleLeftButtonUp();
43             break;
44     }
45 }
```

3.22.2.7 handleMouseMove()

```
void Viewer::handleMouseMove (
            LPARAM lParam )  [private]
```

Handles the mouse move event for panning.

Parameters

lParam	The LPARAM parameter of the message.

Definition at line 59 of file Viewer.cpp.

```
59                                          {
60     if (is_dragging) {
61         int x = static_cast< int >(LOWORD(lParam));
62         int y = static_cast< int >(HIWORD(lParam));
63
64         if (x != last_mouse_pos.x || y != last_mouse_pos.y) {
65             offset_x += (x - last_mouse_pos.x) * zoom_factor;
66             offset_y += (y - last_mouse_pos.y) * zoom_factor;
67             last_mouse_pos.x = x;
68             last_mouse_pos.y = y;
69             needs_repaint = true;
70         }
71     }
72 }
```

3.22.2.8 handleMouseWheel()

```
void Viewer::handleMouseWheel (
            WPARAM wParam )  [private]
```

Handles the mouse wheel event for zooming.

Parameters

wParam	The WPARAM parameter of the message.

Definition at line 49 of file Viewer.cpp.

```
49                                              {
50     if (GET_WHEEL_DELTA_WPARAM(wParam) > 0) {
51         zoom_factor *= 1.1f;
52         needs_repaint = true;
53     } else {
54         zoom_factor /= 1.1f;
55         needs_repaint = true;
56     }
57 }
```

3.22.3 Member Data Documentation

3.22.3.1 needs_repaint

```
bool Viewer::needs_repaint
```

Flag indicating whether the view needs to be repainted

Definition at line 22 of file Viewer.hpp.

The documentation for this class was generated from the following files:

- src/Viewer.hpp
- src/Viewer.cpp

Index

addElement
Group, 22
SVGElement, 96
addPoint
Path, 52
PolyShape, 64
addStop
Gradient, 17
applyTransform
Renderer, 76
applyTransformsOnBrush
Renderer, 77

Circle, 5
Circle, 6
getClass, 7
ColorShape, 8
ColorShape, 9, 10
operator<<, 10

draw
Renderer, 78
drawCircle
Renderer, 79
drawEllipse
Renderer, 80
drawLine
Renderer, 81
drawPath
Renderer, 81
drawPolygon
Renderer, 84
drawPolyline
Renderer, 85
drawRectangle
Renderer, 86
drawText
Renderer, 87

Ell, 11
Ell, 13
getClass, 13
getMaxBound, 13
getMinBound, 14
getRadius, 14
printData, 14
setRadius, 15

getAnchor
Text, 107
getAttribute
Parser, 32
getAttributes
Group, 23
getBrush
Renderer, 88
getClass
Circle, 7
Ell, 13
Gradient, 18
Group, 23
Line, 26
LinearGradient, 29
Path, 53
Plygon, 59
Plyline, 61
PolyShape, 65
RadialGradient, 69
Rect, 72
SVGElement, 97
Text, 107
getColor
Stop, 92
getContent
Text, 107
getDirection
Line, 26
getElements
Group, 23
getFillColor
SVGElement, 97
getFillRule
Path, 53
PolyShape, 65
getFloatAttribute
Parser, 33
getFontSize
Text, 108
getFontStyle
Text, 108
getGradient
SVGElement, 97
GetGradients
Parser, 34
getGradientStops
Parser, 35
getHeight
Rect, 72
ViewBox, 113

getInstance
Parser, 35
Renderer, 90
Viewer, 116
getLength
Line, 27
getMaxBound
Ell, 13
PolyShape, 65
SVGElement, 98
getMinBound
Ell, 14
PolyShape, 65
SVGElement, 98
getOffset
Stop, 92
getOutlineColor
SVGElement, 98
getOutlineThickness
SVGElement, 99
getParent
SVGElement, 99
getPoints
Gradient, 18
Path, 53
PolyShape, 66
getPosition
SVGElement, 99
getRadius
Ell, 14
RadialGradient, 69
Rect, 72
getRoot
Parser, 36
getStops
Gradient, 18
getTransformOrder
Parser, 36
getTransforms
Gradient, 18
SVGElement, 100
getUnits
Gradient, 19
getViewBox
Parser, 37
getViewPort
Parser, 37
getWidth
Rect, 73
ViewBox, 113
getWindowSize
Viewer, 117
getX
ViewBox, 114
getY
ViewBox, 114
Gradient, 15
addStop, 17
getClass, 18
getPoints, 18
getStops, 18
getTransforms, 18
getUnits, 19
Gradient, 17
setTransforms, 19
setUnits, 19
gradients
Parser, 50
Group, 20
addElement, 22
getAttributes, 23
getClass, 23
getElements, 23
Group, 22
printData, 23

handleKeyDown
Viewer, 117
handleKeyEvent
Viewer, 118
handleLeftButtonDown
Viewer, 118
handleMouseEvent
Viewer, 118
handleMouseMove
Viewer, 119
handleMouseWheel
Viewer, 119

Line, 24
getClass, 26
getDirection, 26
getLength, 27
Line, 26
setDirection, 27
LinearGradient, 28
getClass, 29
LinearGradient, 29

needs_repaint
Viewer, 120

operator<<
ColorShape, 10

parseCircle
Parser, 37
parseColor
Parser, 38
parseElements
Parser, 39
parseEllipse
Parser, 41
parseGradient
Parser, 41
parseLine
Parser, 42

parsePath
Parser, 42
parsePathPoints
Parser, 43
parsePoints
Parser, 45
parsePolygon
Parser, 46
parsePolyline
Parser, 46
Parser, 30
getAttribute, 32
getFloatAttribute, 33
GetGradients, 34
getGradientStops, 35
getInstance, 35
getRoot, 36
getTransformOrder, 36
getViewBox, 37
getViewPort, 37
gradients, 50
parseCircle, 37
parseColor, 38
parseElements, 39
parseEllipse, 41
parseGradient, 41
parseLine, 42
parsePath, 42
parsePathPoints, 43
parsePoints, 45
parsePolygon, 46
parsePolyline, 46
Parser, 32
parseRect, 47
parseShape, 48
parseText, 49
printShapesData, 49
parseRect
Parser, 47
parseShape
Parser, 48
parseText
Parser, 49
Path, 50
addPoint, 52
getClass, 53
getFillRule, 53
getPoints, 53
Path, 52
printData, 54
setFillRule, 54
PathPoint, 55
Plygon, 56
getClass, 59
Plygon, 57
Plyline, 59
getClass, 61
Plyline, 61
PolyShape, 62
addPoint, 64
getClass, 65
getFillRule, 65
getMaxBound, 65
getMinBound, 65
getPoints, 66
PolyShape, 64
printData, 66
setFillRule, 66
printData
Ell, 14
Group, 23
Path, 54
PolyShape, 66
Rect, 73
SVGElement, 100
printShapesData
Parser, 49

RadialGradient, 67
getClass, 69
getRadius, 69
RadialGradient, 68
Rect, 70
getClass, 72
getHeight, 72
getRadius, 72
getWidth, 73
printData, 73
Rect, 71
setHeight, 73
setRadius, 74
setWidth, 74
Renderer, 74
applyTransform, 76
applyTransformsOnBrush, 77
draw, 78
drawCircle, 79
drawEllipse, 80
drawLine, 81
drawPath, 81
drawPolygon, 84
drawPolyline, 85
drawRectangle, 86
drawText, 87
getBrush, 88
getInstance, 90

setAnchor
Text, 108
setContent
Text, 109
setDirection
Line, 27
setFillColor
SVGElement, 101
setFillRule
Path, 54

PolyShape, 66
setFontSize
Text, 109
setFontStyle
Text, 109
setGradient
SVGElement, 101
setHeight
Rect, 73
setOutlineColor
SVGElement, 102
setOutlineThickness
SVGElement, 102
setParent
SVGElement, 103
setPosition
SVGElement, 103
setRadius
Ell, 15
Rect, 74
setTransforms
Gradient, 19
SVGElement, 104
setUnits
Gradient, 19
setWidth
Rect, 74
Stop, 90
getColor, 92
getOffset, 92
Stop, 91
SVGElement, 93
addElement, 96
getClass, 97
getFillColor, 97
getGradient, 97
getMaxBound, 98
getMinBound, 98
getOutlineColor, 98
getOutlineThickness, 99
getParent, 99
getPosition, 99
getTransforms, 100
printData, 100
setFillColor, 101
setGradient, 101
setOutlineColor, 102
setOutlineThickness, 102
setParent, 103
setPosition, 103
setTransforms, 104
SVGElement, 95, 96

Text, 105
getAnchor, 107
getClass, 107
getContent, 107
getFontSize, 108
getFontStyle, 108
setAnchor, 108
setContent, 109
setFontSize, 109
setFontStyle, 109
Text, 106

Vector2D
Vector2D< T >, 111
Vector2D< T >, 110
Vector2D, 111
ViewBox, 112
getHeight, 113
getWidth, 113
getX, 114
getY, 114
ViewBox, 112, 113
Viewer, 115
getInstance, 116
getWindowSize, 117
handleKeyDown, 117
handleKeyEvent, 118
handleLeftButtonDown, 118
handleMouseEvent, 118
handleMouseMove, 119
handleMouseWheel, 119
needs_repaint, 120