svg-reader

0.3

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

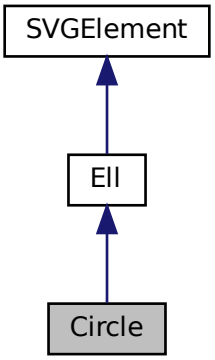
Class Documentation

3.1 Circle Class Reference

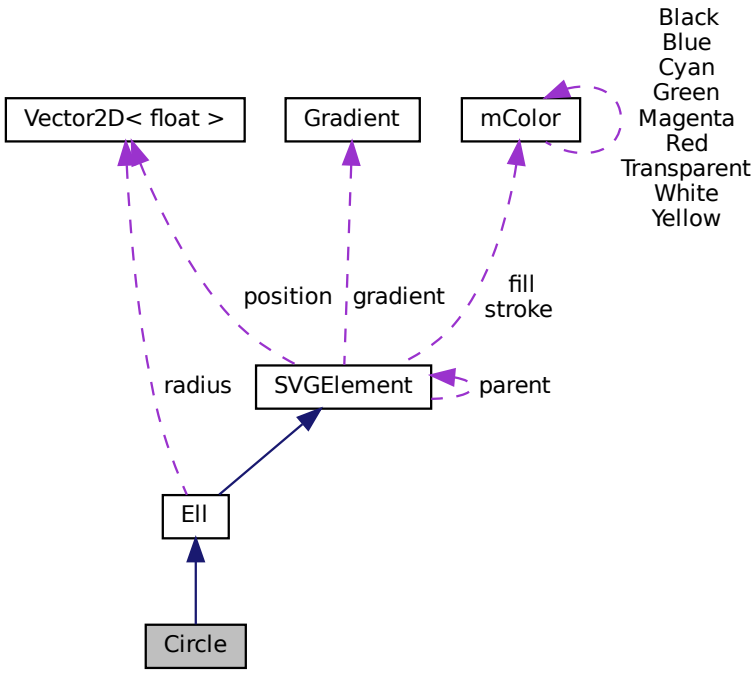
Represents a circle in 2D space.

`#include <Circle.hpp>`

Inheritance diagram for Circle:



Collaboration diagram for Circle:



Public Member Functions

- Circle (float radius, const Vector2Df &center, mColor fill, mColor 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,
            mColor fill,
            mColor 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:

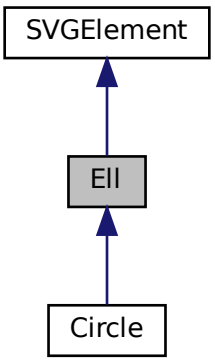
- src/graphics/Circle.hpp
- src/graphics/Circle.cpp

3.2 Ell Class Reference

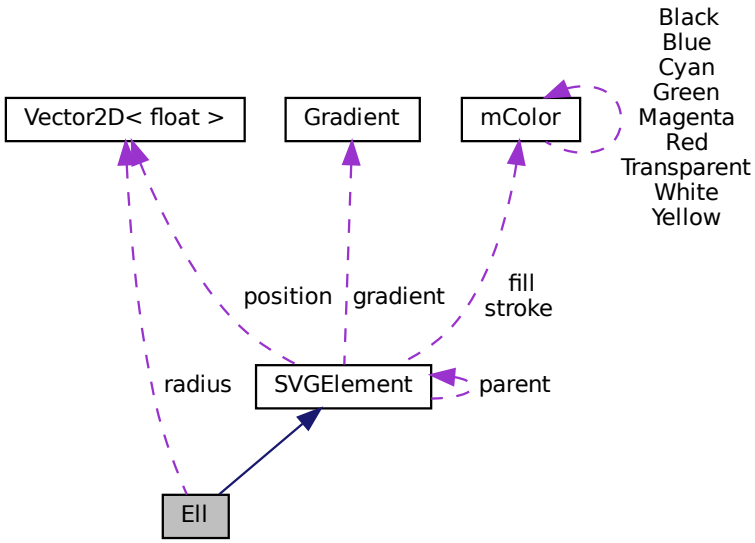
Represents an ellipse in 2D space.

`#include <Ellipse.hpp>`

Inheritance diagram for Ell:



Collaboration diagram for Ell:



Public Member Functions

- Ell (const Vector2Df &radius, const Vector2Df &center, mColor fill, mColor 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

3.2.1 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.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Ell()

```
Ell::Ell (
            const Vector2Df & radius,
            const Vector2Df & center,
            mColor fill,
            mColor 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) {}
```

3.2.3 Member Function Documentation

3.2.3.1 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"; }
```

3.2.3.2 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 }
```

3.2.3.3 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 }
```

3.2.3.4 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; }
```

3.2.3.5 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 }
```

3.2.3.6 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:

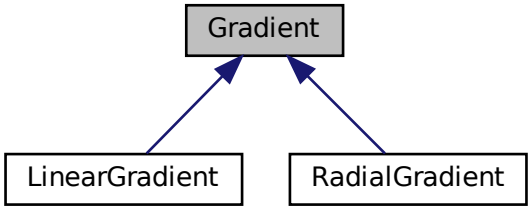
- src/graphics/Ellipse.hpp
- src/graphics/Ellipse.cpp

3.3 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.

3.3.1 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.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 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) {}
```

3.3.3 Member Function Documentation

3.3.3.1 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); }
```

3.3.3.2 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.

3.3.3.3 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; }
```

3.3.3.4 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; }
```

3.3.3.5 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 }
```

3.3.3.6 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; }
```

3.3.3.7 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 }
```

3.3.3.8 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:

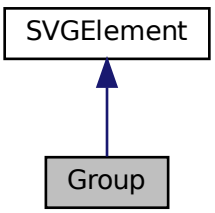
- src/graphics/Gradient.hpp
- src/graphics/Gradient.cpp

3.4 Group Class Reference

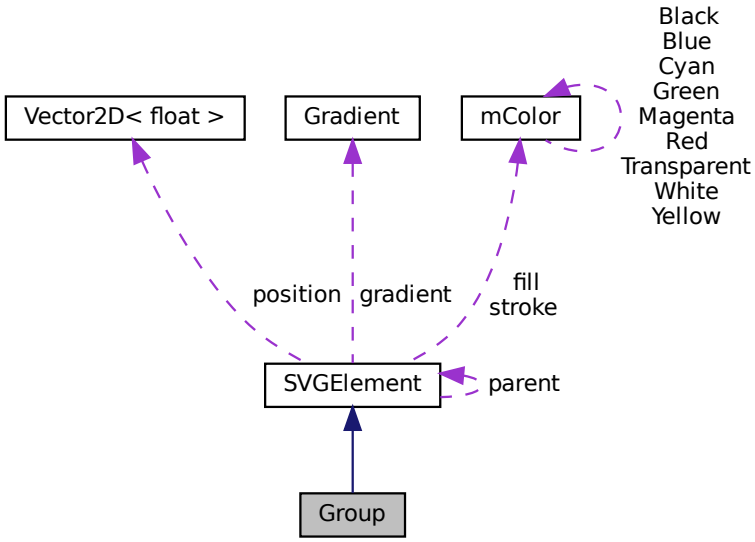
A composite class that contains a vector of shape pointers (polymorphic).

`#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

3.4.1 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 19 of file Group.hpp.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 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) {}
```

3.4.3 Member Function Documentation

3.4.3.1 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 }
```

3.4.3.2 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; }
```

3.4.3.3 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"; }
```

3.4.3.4 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; }
```

3.4.3.5 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:

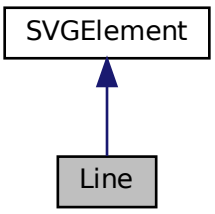
- src/graphics/Group.hpp
- src/graphics/Group.cpp

3.5 Line Class Reference

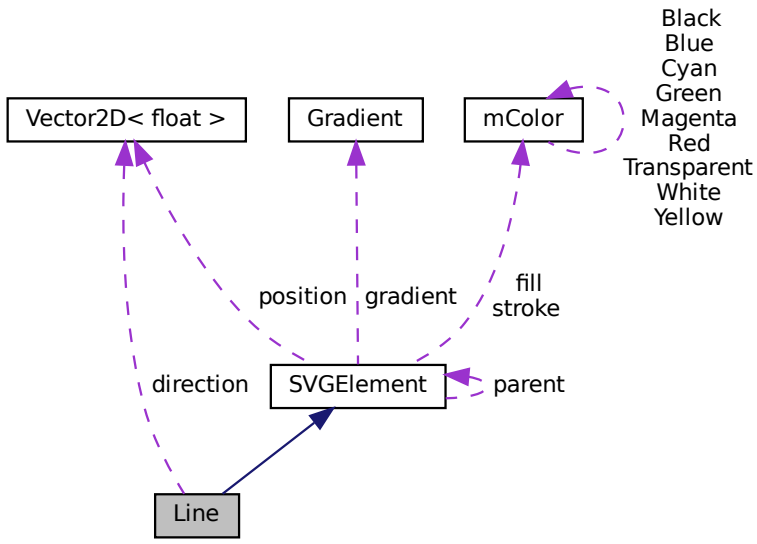
Represents a line in 2D space.

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

Inheritance diagram for Line:



Collaboration diagram for Line:



Public Member Functions

- Line (const Vector2Df &point1, const Vector2Df &point2, mColor 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

3.5.1 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.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Line()

```
Line::Line (
            const Vector2Df & point1,
            const Vector2Df & point2,
            mColor 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(mColor::Transparent, stroke, stroke_width, point1),
8       direction(point2) {}
```

3.5.3 Member Function Documentation

3.5.3.1 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"; }
```

3.5.3.2 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; }
```

3.5.3.3 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 }
```

3.5.3.4 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:

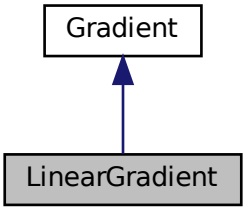
- src/graphics/Line.hpp
- src/graphics/Line.cpp

3.6 LinearGradient Class Reference

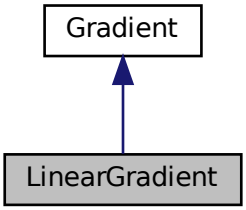
A class that represents a linear gradient.

`#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.

3.6.1 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.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 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) {}
```

3.6.3 Member Function Documentation

3.6.3.1 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:

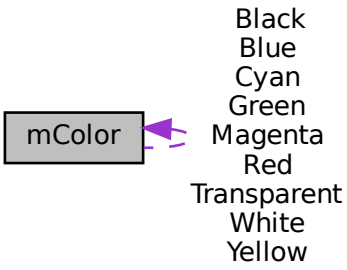
- src/graphics/LinearGradient.hpp
- src/graphics/LinearGradient.cpp

3.7 mColor Class Reference

Utility class for manipulating RGBA mColors.

`#include <Color.hpp>`

Collaboration diagram for mColor:



Public Member Functions

- mColor ()

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

  Construct the mColor from its 4 RGBA components.
- mColor (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 mColor Black
  Black predefined color.
- static const mColor White
  White predefined color.
- static const mColor Red
  Red predefined color.
- static const mColor Green
  Green predefined color.
- static const mColor Blue
  Blue predefined color.
- static const mColor Yellow
  Yellow predefined color.
- static const mColor Magenta
  Magenta predefined color.
- static const mColor Cyan
  Cyan predefined color.
- static const mColor Transparent
  Transparent (black) predefined color.

Friends

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

3.7.1 Detailed Description

Utility class for manipulating RGBA mColors.

Definition at line 11 of file Color.hpp.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 mColor() [1/3]

```
mColor::mColor ( )
```

Default constructor.

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

Definition at line 14 of file Color.cpp.

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

3.7.2.2 mColor() [2/3]

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

Construct the mColor 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 Color.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 }
```

3.7.2.3 mColor() [3/3]

```
mColor::mColor (
            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 Color.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) {}
```

3.7.3 Friends And Related Function Documentation

3.7.3.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const mColor & 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 Color.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:

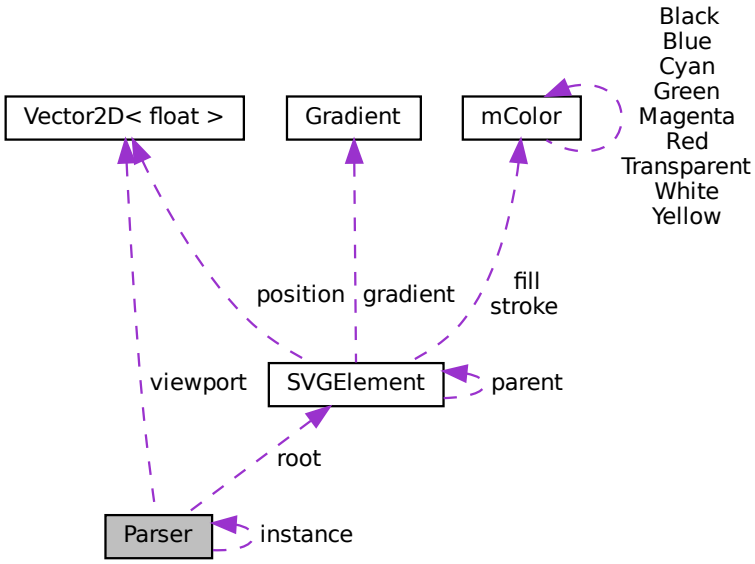
- src/graphics/Color.hpp
- src/graphics/Color.cpp

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.
- std::pair< Vector2Df, Vector2Df > 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.
- mColor 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 mColor &stroke_color, float stroke_width)

  Parses the line element.
- Rect * parseRect (rapidxml::xml_node<> *node, const mColor &fill_color, const mColor &stroke_color, float stroke_width)

  Parses the rect element.

- class Plyline * parsePolyline (rapidxml::xml_node<> *node, const mColor &fill_color, const mColor &stroke_color, float stroke_width)

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

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

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

  Parses the ellipse element.
- Path * parsePath (rapidxml::xml_node<> *node, const mColor &fill_color, const mColor &stroke_color, float stroke_width)

  Parses the path element.
- Text * parseText (rapidxml::xml_node<> *node, const mColor &fill_color, const mColor &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
- std::pair< Vector2Df, Vector2Df > 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 165 of file Parser.cpp.

```
165                                         {
166     root = parseElements(file_name);
167 }
```

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 296 of file Parser.cpp.

```
296                                                                         {
297     if (name == "text") return removeExtraSpaces(node->value());
298     std::string result;
299     if (node->first_attribute(name.c_str()) == NULL) {
300         if (name == "fill" || name == "stop-color")
301             result = "black";
302         else if (name == "stroke" || name == "transform" || name == "rotate" ||
303                  name == "font-style")
304             result = "none";
305         else if (name == "text-anchor")
306             result = "start";
307         else if (name == "fill-rule")
308             result = "nonzero";
309         else if (name == "gradientUnits")
310             result = "objectBoundingBox";
311     } else {
312         result = node->first_attribute(name.c_str())->value();
313     }
314     return result;
315 }
```

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 317 of file Parser.cpp.

```
317                                                                                 {
318     float result;
319     if (node->first_attribute(name.c_str()) == NULL) {
320         if (std::string(node->name()).find("Gradient") != std::string::npos) {
321             if (name == "x1" || name == "y1" || name == "fr")
322                 result = 0;
323             else if (name == "cx" || name == "cy")
324                 result = name == "cx" ? 0.5 * this->viewbox.second.x
325                                       : 0.5 * this->viewbox.second.y;
326             else if (name == "r") {
327                 result = sqrt((pow(this->viewbox.second.x, 2) +
328                                pow(this->viewbox.second.y, 2)) /
329                               2) /
330                          2;
331             } else if (name == "fx" || name == "fy")
332                 result = name == "fx" ? getFloatAttribute(node, "cx")
333                                       : getFloatAttribute(node, "cy");
334             else
335                 result = name == "x2" ? this->viewbox.second.x
336                                       : this->viewbox.second.y;
337         } else {
338             if (name == "stroke-width" || name == "stroke-opacity" ||
339                 name == "fill-opacity" || name == "opacity" ||
340                 name == "stop-opacity")
341                 result = 1;
342             else
343                 result = 0;
344         }
345     } else {
346         if (name == "width" || name == "height") {
347             std::string value = node->first_attribute(name.c_str())->value();
348             if (value.find("%") != std::string::npos) {
349                 result = std::stof(value.substr(0, value.find("%"))) *
350                          this->viewbox.second.x / 100;
351             } else if (value.find("pt") != std::string::npos) {
352                 result = std::stof(value.substr(0, value.find("pt"))) * 1.33;
353             } else {
354                 result = std::stof(value);
355             }
356         } else
357             result = std::stof(node->first_attribute(name.c_str())->value());
358     }
359     return result;
360 }
```

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 428 of file Parser.cpp.

```
428                                                 {
429     rapidxml::xml_node<> *gradient_node = node->first_node();
430     while (gradient_node) {
431         if (std::string(gradient_node->name()).find("Gradient") !=
432             std::string::npos) {
433             Gradient *gradient;
434             std::string id = getAttribute(gradient_node, "id");
435             std::string units = getAttribute(gradient_node, "gradientUnits");
436             std::vector< Stop > stops = getGradientStops(gradient_node);
437             std::string href = getAttribute(gradient_node, "xlink:href");
438             int pos = href.find("#");
439             if (pos != std::string::npos) {
440                 href = href.substr(pos + 1);
441             }
442             if (std::string(gradient_node->name()).find("linear") !=
443                 std::string::npos) {
444                 float x1 = getFloatAttribute(gradient_node, "x1");
445                 float y1 = getFloatAttribute(gradient_node, "y1");
446                 float x2 = getFloatAttribute(gradient_node, "x2");
447                 float y2 = getFloatAttribute(gradient_node, "y2");
448                 std::pair< Vector2Df, Vector2Df > points = {{x1, y1}, {x2, y2}};
449                 gradient = new LinearGradient(stops, points, units);
450                 if (this->gradients.find(id) == this->gradients.end())
451                     this->gradients[id] = gradient;
452             } else if (std::string(gradient_node->name()).find("radial") !=
453                        std::string::npos) {
454                 float cx = getFloatAttribute(gradient_node, "cx");
455                 float cy = getFloatAttribute(gradient_node, "cy");
456                 float fx = getFloatAttribute(gradient_node, "fx");
457                 float fy = getFloatAttribute(gradient_node, "fy");
458                 float r = getFloatAttribute(gradient_node, "r");
459                 float fr = getFloatAttribute(gradient_node, "fr");
460                 std::pair< Vector2Df, Vector2Df > points = {{cx, cy}, {fx, fy}};
461                 Vector2Df radius(r, fr);
462                 gradient = new RadialGradient(stops, points, radius, units);
463                 if (this->gradients.find(id) == this->gradients.end())
464                     this->gradients[id] = gradient;
465             }
466             if (href != "") {
467                 for (auto stop : parseGradient(href)->getStops()) {
468                     gradient->addStop(stop);
469                 }
470             }
471             if (gradient != NULL)
472                 gradient->setTransforms(getTransformOrder(gradient_node));
473         }
474         gradient_node = gradient_node->next_sibling();
475     }
476 }
```

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 412 of file Parser.cpp.

```
412                                                                 {
413     std::vector< Stop > stops;
414     rapidxml::xml_node<> *stop_node = node->first_node();
```

```
415     while (stop_node) {
416         if (std::string(stop_node->name()) == "stop") {
417             std::string id = "";
418             mColor color = parseColor(stop_node, "stop-color", id);
419             float offset = getFloatAttribute(stop_node, "offset");
420             if (offset > 1) offset /= 100;
421             stops.push_back(Stop(color, offset));
422         }
423         stop_node = stop_node->next_sibling();
424     }
425     return stops;
426 }
```

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 158 of file Parser.cpp.

```
158                                                     {
159     if (instance == nullptr) {
160         instance = new Parser(file_name);
161     }
162     return instance;
163 }
```

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 169 of file Parser.cpp.

```
169 { 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 637 of file Parser.cpp.

```
638                                 {
639     std::string transform_tag;
640     if (std::string(node->name()).find("Gradient") != std::string::npos)
641         transform_tag = getAttribute(node, "gradientTransform");
642     else
643         transform_tag = getAttribute(node, "transform");
644     std::vector< std::string > order;
645     std::stringstream ss(transform_tag);
646     std::string type;
647     while (ss » type) {
648         if (type.find("translate") != std::string::npos ||
649             type.find("scale") != std::string::npos ||
650             type.find("rotate") != std::string::npos ||
651             type.find("matrix") != std::string::npos) {
652             while (type.find(")") == std::string::npos) {
653                 std::string temp;
654                 ss » temp;
655                 type += " " + temp;
656             }
657             std::string temp = type.substr(0, type.find("(") + 1);
658             temp.erase(std::remove(temp.begin(), temp.end(), ' '), temp.end());
659             type.erase(0, type.find("(") + 1);
660             type = temp + type;
661             order.push_back(type);
662         }
663     }
664     return order;
665 }
```

3.8.3.8 getViewBox()

```
std::pair< Vector2Df, Vector2Df > Parser::getViewBox ( ) const
```

Gets the viewbox of the SVG file.

Returns

The viewbox of the SVG file.

Definition at line 830 of file Parser.cpp.

```
830 { 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 832 of file Parser.cpp.

```
832 { return viewport; }
```

3.8.3.10 parseCircle()

```
Circle * Parser::parseCircle (
            rapidxml::xml_node<> * node,
            const mColor & fill_color,
            const mColor & 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 732 of file Parser.cpp.

```
734                                                                                 {
735     float cx = getFloatAttribute(node, "cx");
736     float cy = getFloatAttribute(node, "cy");
737     float radius = getFloatAttribute(node, "r");
738     Circle *shape = new Circle(radius, Vector2Df(cx, cy), fill_color,
739                                stroke_color, stroke_width);
740     return shape;
741 }
```

3.8.3.11 parseColor()

```
mColor 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 362 of file Parser.cpp.

```
363                                     {
```

```
364     std::string color = getAttribute(node, name);
365     color.erase(std::remove(color.begin(), color.end(), ' '), color.end());
366     if (color.find("url") == std::string::npos) {
367         for (auto &c : color) c = tolower(c);
368     }
369     if (color == "none")
370         return mColor::Transparent;
371     else {
372         mColor result;
373         if (color.find("url") != std::string::npos) {
374             if (color.find("'") != std::string::npos) {
375                 id = color.substr(color.find("'") + 1);
376                 id.erase(id.find("'"));
377                 id.erase(id.find("#"), 1);
378             } else {
379                 id = color.substr(color.find("#") + 1);
380                 id.erase(id.find(")"));
381             }
382             result = mColor::Transparent;
383         } else if (color.find("#") != std::string::npos) {
384             result = getHexColor(color);
385         } else if (color.find("rgb") != std::string::npos) {
386             result = getRgbColor(color);
387         } else {
388             auto color_code = color_map.find(color);
389             if (color_code == color_map.end()) {
390                 std::cout « "Color " « color « " not found" « std::endl;
391                 exit(-1);
392             }
393             result = color_code->second;
394         }
395         if (name == "stop-color")
396             result.a = result.a * getFloatAttribute(node, "stop-opacity");
397         else
398             result.a = result.a * getFloatAttribute(node, name + "-opacity") *
399                        getFloatAttribute(node, "opacity");
400         return result;
401     }
402 }
```

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 181 of file Parser.cpp.

```
181                                                 {
182     rapidxml::xml_document<> doc;
183     std::ifstream file(file_name);
184     std::vector< char > buffer((std::istreambuf_iterator< char >(file)),
185                                std::istreambuf_iterator< char >());
186     buffer.push_back('\0');
187     doc.parse< 0 >(&buffer[0]);
188
189     rapidxml::xml_node<> *svg = doc.first_node();
190     viewport.x = getFloatAttribute(svg, "width");
191     viewport.y = getFloatAttribute(svg, "height");
192     std::string viewbox = getAttribute(svg, "viewBox");
193     if (viewbox != "") {
194         std::stringstream ss(viewbox);
```

```
195         ss » this->viewbox.first.x » this->viewbox.first.y »
196             this->viewbox.second.x » this->viewbox.second.y;
197     }
198     rapidxml::xml_node<> *node = svg->first_node();
199     rapidxml::xml_node<> *prev = NULL;
200
201     SVGElement *root = new Group();
202     SVGElement *current = root;
203
204     while (node) {
205         if (std::string(node->name()) == "defs") {
206             GetGradients(node);
207             prev = node;
208             node = node->next_sibling();
209         } else if (std::string(node->name()) == "g") {
210             Group *group = dynamic_cast< Group * >(current);
211             for (auto group_attribute : group->getAttributes()) {
212                 bool found = false;
213                 for (auto attribute = node->first_attribute(); attribute;
214                      attribute = attribute->next_attribute()) {
215                     if (std::string(attribute->name()) ==
216                         group_attribute.first) {
217                         if (group_attribute.first == "opacity") {
218                             std::string opacity = std::to_string(
219                                 std::stof(attribute->value()) *
220                                 std::stof(group_attribute.second));
221                             char *value = doc.allocate_string(opacity.c_str());
222                             attribute->value(value);
223                         }
224                         found = true;
225                         break;
226                     }
227                 }
228                 if (!found && group_attribute.first != "transform") {
229                     char *name =
230                         doc.allocate_string(group_attribute.first.c_str());
231                     char *value =
232                         doc.allocate_string(group_attribute.second.c_str());
233                     rapidxml::xml_attribute<> *new_attribute =
234                         doc.allocate_attribute(name, value);
235                     node->append_attribute(new_attribute);
236                 }
237             }
238             Group *new_group = new Group(xmlToString(node->first_attribute()));
239             new_group->setTransforms(getTransformOrder(node));
240             current->addElement(new_group);
241             current = new_group;
242             prev = node;
243             node = node->first_node();
244         } else {
245             Group *group = dynamic_cast< Group * >(current);
246             for (auto group_attribute : group->getAttributes()) {
247                 bool found = false;
248                 for (auto attribute = node->first_attribute(); attribute;
249                      attribute = attribute->next_attribute()) {
250                     if (std::string(attribute->name()) ==
251                         group_attribute.first) {
252                         if (group_attribute.first == "opacity") {
253                             std::string opacity = std::to_string(
254                                 std::stof(attribute->value()) *
255                                 std::stof(group_attribute.second));
256                             char *value = doc.allocate_string(opacity.c_str());
257                             attribute->value(value);
258                         }
259                         found = true;
260                         break;
261                     }
262                 }
263                 if (!found && group_attribute.first != "transform") {
264                     char *name =
265                         doc.allocate_string(group_attribute.first.c_str());
266                     char *value =
267                         doc.allocate_string(group_attribute.second.c_str());
268                     rapidxml::xml_attribute<> *new_attribute =
269                         doc.allocate_attribute(name, value);
270                     node->append_attribute(new_attribute);
271                 }
272             }
273             SVGElement *shape = parseShape(node);
274             if (shape != NULL) current->addElement(shape);
275             prev = node;
276             node = node->next_sibling();
277         }
278         if (node == NULL && current != root) {
279             while (prev->parent()->next_sibling() == NULL) {
280                 current = current->getParent();
281                 prev = prev->parent();
```

```
282                 if (prev == svg) {
283                     break;
284                 }
285             }
286             if (prev == svg) {
287                 break;
288             }
289             current = current->getParent();
290             node = prev->parent()->next_sibling();
291         }
292     }
293     return root;
294 }
```

3.8.3.13 parseEllipse()

```
Ell * Parser::parseEllipse (
            rapidxml::xml_node<> * node,
            const mColor & fill_color,
            const mColor & 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 743 of file Parser.cpp.

```
744                                                                                 {
745     float radius_x = getFloatAttribute(node, "rx");
746     float radius_y = getFloatAttribute(node, "ry");
747     float cx = getFloatAttribute(node, "cx");
748     float cy = getFloatAttribute(node, "cy");
749     Ell *shape = new Ell(Vector2Df(radius_x, radius_y), Vector2Df(cx, cy),
750                          fill_color, stroke_color, stroke_width);
751     return shape;
752 }
```

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 404 of file Parser.cpp.

```
404                                                 {
405     if (gradients.find(id) == gradients.end()) {
406         std::cout « "Gradient " « id « " not found" « std::endl;
407         exit(-1);
408     }
409     return gradients.at(id);
410 }
```

3.8.3.15 parseLine()

```
Line * Parser::parseLine (
            rapidxml::xml_node<> * node,
            const mColor & 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 710 of file Parser.cpp.

```
711                                                 {
712     Line *shape = new Line(
713         Vector2Df(getFloatAttribute(node, "x1"), getFloatAttribute(node, "y1")),
714         Vector2Df(getFloatAttribute(node, "x2"), getFloatAttribute(node, "y2")),
715         stroke_color, stroke_width);
716     return shape;
717 }
```

3.8.3.16 parsePath()

```
Path * Parser::parsePath (
            rapidxml::xml_node<> * node,
            const mColor & fill_color,
            const mColor & 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 807 of file Parser.cpp.

```
808                                                                                 {
809     Path *shape = new Path(fill_color, stroke_color, stroke_width);
810     std::vector< PathPoint > points = parsePathPoints(node);
811     for (auto point : points) {
812         shape->addPoint(point);
813     }
814     std::string fill_rule = getAttribute(node, "fill-rule");
815     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
816                     fill_rule.end());
817     shape->setFillRule(fill_rule);
818     return shape;
819 }
```

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 494 of file Parser.cpp.

```
494                                                                                 {
495     std::vector< PathPoint > points;
496     std::string path_string = getAttribute(node, "d");
497
498     formatSvgPathString(path_string);
499
500     std::stringstream ss(path_string);
501     std::string element;
502     PathPoint pPoint{{0, 0}, 'M'};
503     while (ss » element) {
504         if (std::isalpha(element[0])) {
505             pPoint.tc = element[0];
506             if (tolower(pPoint.tc) == 'm' || tolower(pPoint.tc) == 'l' ||
507                 tolower(pPoint.tc) == 'c' || tolower(pPoint.tc) == 's' ||
508                 tolower(pPoint.tc) == 'q' || tolower(pPoint.tc) == 't')
509                 ss » pPoint.point.x » pPoint.point.y;
510             else if (tolower(pPoint.tc) == 'h') {
511                 ss » pPoint.point.x;
512                 pPoint.point.y = 0;
```

```
513             } else if (tolower(pPoint.tc) == 'v') {
514                 ss » pPoint.point.y;
515                 pPoint.point.x = 0;
516             } else if (tolower(pPoint.tc) == 'a') {
517                 ss » pPoint.radius.x » pPoint.radius.y;
518                 ss » pPoint.x_axis_rotation;
519                 ss » pPoint.large_arc_flag » pPoint.sweep_flag;
520                 ss » pPoint.point.x » pPoint.point.y;
521             }
522         } else {
523             if (tolower(pPoint.tc) == 'm' || tolower(pPoint.tc) == 'l' ||
524                 tolower(pPoint.tc) == 'c' || tolower(pPoint.tc) == 's' ||
525                 tolower(pPoint.tc) == 'q' || tolower(pPoint.tc) == 't') {
526                 if (tolower(pPoint.tc) == 'm') pPoint.tc = 'L';
527                 pPoint.point.x = std::stof(element);
528                 ss » pPoint.point.y;
529             } else if (tolower(pPoint.tc) == 'h') {
530                 pPoint.point.x = std::stof(element);
531                 pPoint.point.y = 0;
532             } else if (tolower(pPoint.tc) == 'v') {
533                 pPoint.point.y = std::stof(element);
534                 pPoint.point.x = 0;
535             } else if (tolower(pPoint.tc) == 'a') {
536                 pPoint.radius.x = std::stof(element);
537                 ss » pPoint.radius.y;
538                 ss » pPoint.x_axis_rotation;
539                 ss » pPoint.large_arc_flag » pPoint.sweep_flag;
540                 ss » pPoint.point.x » pPoint.point.y;
541             }
542         }
543         points.push_back(pPoint);
544     }
545
546     std::vector< PathPoint > handle_points;
547
548     Vector2Df first_point{0, 0}, cur_point{0, 0};
549     int n = points.size();
550     for (int i = 0; i < n; i++) {
551         if (tolower(points[i].tc) == 'm') {
552             first_point = points[i].point;
553             if (points[i].tc == 'm') {
554                 first_point.x = cur_point.x + points[i].point.x;
555                 first_point.y = cur_point.y + points[i].point.y;
556             }
557             cur_point = first_point;
558             handle_points.push_back({first_point, 'm'});
559         } else if (tolower(points[i].tc) == 'l' ||
560                    tolower(points[i].tc) == 't') {
561             Vector2Df end_point{cur_point.x + points[i].point.x,
562                                 cur_point.y + points[i].point.y};
563             if (points[i].tc == 'L' || points[i].tc == 'T')
564                 end_point = points[i].point;
565             cur_point = end_point;
566             char TC = tolower(points[i].tc);
567             handle_points.push_back({end_point, TC});
568         } else if (tolower(points[i].tc) == 'h') {
569             Vector2Df end_point{cur_point.x + points[i].point.x, cur_point.y};
570             if (points[i].tc == 'H')
571                 end_point = Vector2Df{points[i].point.x, cur_point.y};
572             cur_point = end_point;
573             handle_points.push_back({end_point, 'h'});
574         } else if (tolower(points[i].tc) == 'v') {
575             Vector2Df end_point{cur_point.x, cur_point.y + points[i].point.y};
576             if (points[i].tc == 'V')
577                 end_point = Vector2Df{cur_point.x, points[i].point.y};
578             cur_point = end_point;
579             handle_points.push_back({end_point, 'v'});
580         } else if (tolower(points[i].tc) == 'c') {
581             if (i + 2 < n) {
582                 Vector2Df control_point1 =
583                     Vector2Df{cur_point.x + points[i].point.x,
584                               cur_point.y + points[i].point.y};
585                 Vector2Df control_point2 =
586                     Vector2Df{cur_point.x + points[i + 1].point.x,
587                               cur_point.y + points[i + 1].point.y};
588                 Vector2Df control_point3 =
589                     Vector2Df{cur_point.x + points[i + 2].point.x,
590                               cur_point.y + points[i + 2].point.y};
591                 if (points[i].tc == 'C') {
592                     control_point1 = points[i].point;
593                     control_point2 = points[i + 1].point;
594                     control_point3 = points[i + 2].point;
595                 }
596                 i += 2;
597                 cur_point = control_point3;
598                 handle_points.push_back({control_point1, 'c'});
599                 handle_points.push_back({control_point2, 'c'});
```

```
600                 handle_points.push_back({control_point3, 'c'});
601             }
602         } else if (tolower(points[i].tc) == 'z') {
603             cur_point = first_point;
604             handle_points.push_back({first_point, 'z'});
605         } else if (tolower(points[i].tc) == 's' ||
606                    tolower(points[i].tc) == 'q') {
607             if (i + 1 < n) {
608                 Vector2Df control_point1 =
609                     Vector2Df{cur_point.x + points[i].point.x,
610                               cur_point.y + points[i].point.y};
611                 Vector2Df control_point2 =
612                     Vector2Df{cur_point.x + points[i + 1].point.x,
613                               cur_point.y + points[i + 1].point.y};
614                 if (points[i].tc == 'S' || points[i].tc == 'Q') {
615                     control_point1 = points[i].point;
616                     control_point2 = points[i + 1].point;
617                 }
618                 i += 1;
619                 cur_point = control_point2;
620                 char TC = tolower(points[i].tc);
621                 handle_points.push_back({control_point1, TC});
622                 handle_points.push_back({control_point2, TC});
623             }
624         } else if (tolower(points[i].tc) == 'a') {
625             Vector2Df end_point{cur_point.x + points[i].point.x,
626                                 cur_point.y + points[i].point.y};
627             if (points[i].tc == 'A') end_point = points[i].point;
628             handle_points.push_back(
629                 {end_point, 'a', points[i].radius, points[i].x_axis_rotation,
630                  points[i].large_arc_flag, points[i].sweep_flag});
631             cur_point = end_point;
632         }
633     }
634     return handle_points;
635 }
```

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 478 of file Parser.cpp.

```
478                                                                     {
479     std::vector< Vector2Df > points;
480     std::string points_string = getAttribute(node, "points");
481
482     std::stringstream ss(points_string);
483     float x, y;
484
485     while (ss >> x) {
486         if (ss.peek() == ',') ss.ignore();
487         ss >> y;
488         points.push_back(Vector2Df(x, y));
489     }
490
491     return points;
492 }
```

3.8.3.19 parsePolygon()

```
Plygon * Parser::parsePolygon (
            rapidxml::xml_node<> * node,
            const mColor & fill_color,
            const mColor & 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 754 of file Parser.cpp.

```
756                                                                                 {
757     Plygon *shape = new Plygon(fill_color, stroke_color, stroke_width);
758     std::vector< Vector2Df > points = parsePoints(node);
759     for (auto point : points) {
760         shape->addPoint(point);
761     }
762     std::string fill_rule = getAttribute(node, "fill-rule");
763     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
764                     fill_rule.end());
765     shape->setFillRule(fill_rule);
766     return shape;
767 }
```

3.8.3.20 parsePolyline()

```
Plyline * Parser::parsePolyline (
            rapidxml::xml_node<> * node,
            const mColor & fill_color,
            const mColor & 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 769 of file Parser.cpp.

```
771                                                                                 {
772     Pyline *shape = new Pyline(fill_color, stroke_color, stroke_width);
773     std::vector< Vector2Df > points = parsePoints(node);
774     for (auto point : points) {
775         shape->addPoint(point);
776     }
777     std::string fill_rule = getAttribute(node, "fill-rule");
778     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
779                     fill_rule.end());
780     shape->setFillRule(fill_rule);
781     return shape;
782 }
```

3.8.3.21 parseRect()

```
Rect * Parser::parseRect (
            rapidxml::xml_node<> * node,
            const mColor & fill_color,
            const mColor & 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 719 of file Parser.cpp.

```
720                                                                             {
721     float x = getFloatAttribute(node, "x");
722     float y = getFloatAttribute(node, "y");
723     float rx = getFloatAttribute(node, "rx");
724     float ry = getFloatAttribute(node, "ry");
725     Rect *shape =
726         new Rect(getFloatAttribute(node, "width"),
727                  getFloatAttribute(node, "height"), Vector2Df(x, y),
728                  Vector2Df(rx, ry), fill_color, stroke_color, stroke_width);
729     return shape;
730 }
```

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 667 of file Parser.cpp.

```
667                                                     {
668     SVGElement *shape = NULL;
669     std::string type = node->name();
670     std::string id = "";
671     mColor stroke_color = parseColor(node, "stroke", id);
672     mColor fill_color = parseColor(node, "fill", id);
673     float stroke_width = getFloatAttribute(node, "stroke-width");
674     if (type == "line") {
675         shape = parseLine(node, stroke_color, stroke_width);
676     } else if (type == "rect") {
677         shape = parseRect(node, fill_color, stroke_color, stroke_width);
678     } else if (type == "circle") {
679         shape = parseCircle(node, fill_color, stroke_color, stroke_width);
680     } else if (type == "ellipse") {
681         shape = parseEllipse(node, fill_color, stroke_color, stroke_width);
682     } else if (type == "polygon") {
683         shape = parsePolygon(node, fill_color, stroke_color, stroke_width);
684     } else if (type == "polyline") {
685         shape = parsePolyline(node, fill_color, stroke_color, stroke_width);
686     } else if (type == "path") {
687         shape = parsePath(node, fill_color, stroke_color, stroke_width);
688     } else if (type == "text") {
689         shape = parseText(node, fill_color, stroke_color, stroke_width);
690     }
691     if (shape != NULL) {
692         if (type == "text") {
693             float dx = getFloatAttribute(node, "dx");
694             float dy = getFloatAttribute(node, "dy");
695             std::string transform = "translate(" + std::to_string(dx) + " " +
696                                     std::to_string(dy) + ")";
697             std::vector< std::string > transform_order =
698                 getTransformOrder(node);
699             transform_order.push_back(transform);
700             shape->setTransforms(transform_order);
701         } else
702             shape->setTransforms(getTransformOrder(node));
703         if (id != "") {
704             shape->setGradient(parseGradient(id));
705         }
706     }
707     return shape;
708 }
```

3.8.3.23 parseText()

```
Text * Parser::parseText (
            rapidxml::xml_node<> * node,
            const mColor & fill_color,
            const mColor & 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 784 of file Parser.cpp.

```
785                                                                                 {
786     float x = getFloatAttribute(node, "x");
787     float y = getFloatAttribute(node, "y");
788     float font_size = getFloatAttribute(node, "font-size");
789     std::string text = getAttribute(node, "text");
790 
791     Text *shape =
792         new Text(Vector2Df(x - (font_size * 6.6 / 40),
793                            y - font_size + (font_size * 4.4 / 40)),
794                  text, font_size, fill_color, stroke_color, stroke_width);
795 
796     std::string anchor = getAttribute(node, "text-anchor");
797     anchor.erase(std::remove(anchor.begin(), anchor.end(), ' '), anchor.end());
798     shape->setAnchor(anchor);
799 
800     std::string style = getAttribute(node, "font-style");
801     style.erase(std::remove(style.begin(), style.end(), ' '), style.end());
802     shape->setFontStyle(style);
803 
804     return shape;
805 }
```

3.8.3.24 printShapesData()

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

Prints the data of the shapes.

Note

This function is used for debugging.

Definition at line 828 of file Parser.cpp.

```
828 { 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 277 of file Parser.hpp.

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

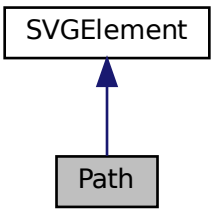
- src/Parser.hpp
- src/Parser.cpp

3.9 Path Class Reference

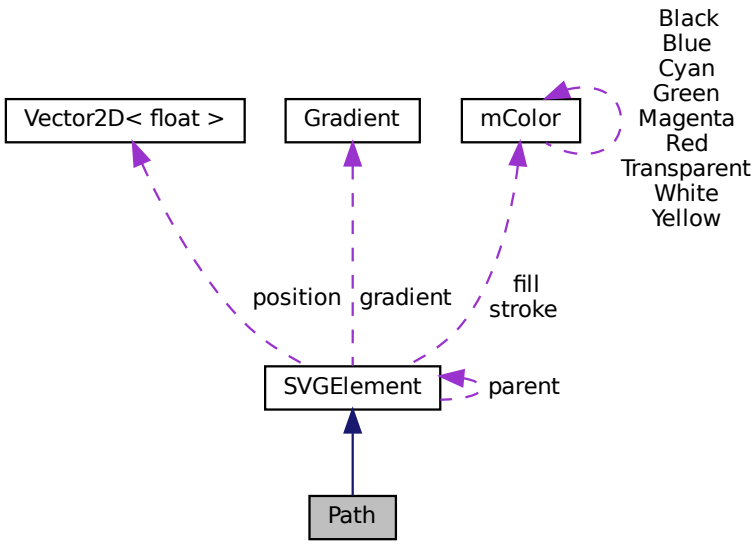
Represents a path element in 2D space.

`#include <Path.hpp>`

Inheritance diagram for Path:



Collaboration diagram for Path:



Public Member Functions

- Path (const mColor &fill, const mColor &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 mColor & fill,
            const mColor & 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:

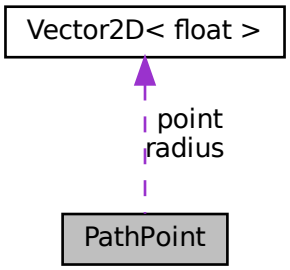
- src/graphics/Path.hpp
- src/graphics/Path.cpp

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**
- char **tc**
- Vector2Df **radius** {0, 0}
- float **x_axis_rotation** = 0.f
- bool **large_arc_flag** = false
- bool **sweep_flag** = false

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:

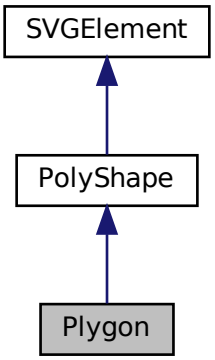
- src/graphics/Path.hpp

3.11 Plygon Class Reference

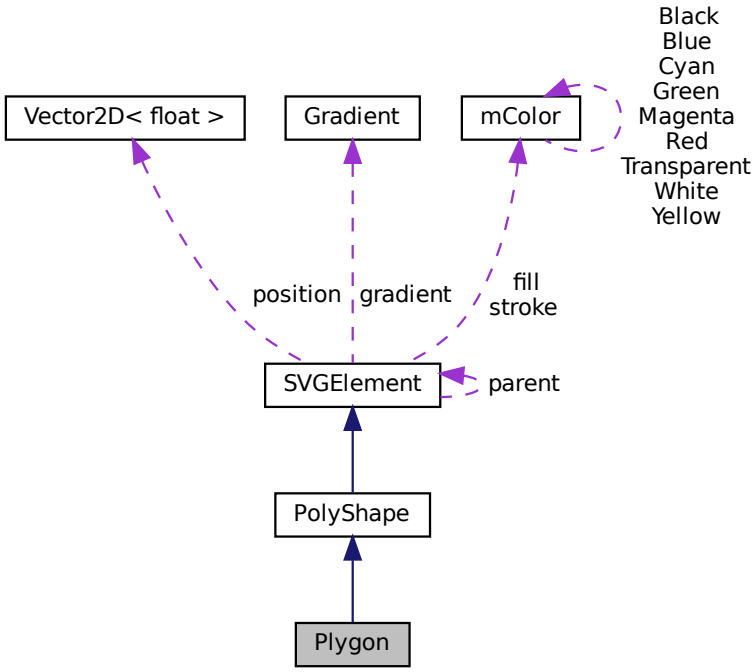
Represents a polygon in 2D space.

`#include <Polygon.hpp>`

Inheritance diagram for Plygon:



Collaboration diagram for Plygon:



Public Member Functions

- Plygon (mColor fill, mColor 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 (
            mColor fill,
            mColor 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:

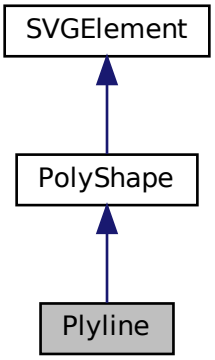
- src/graphics/Polygon.hpp
- src/graphics/Polygon.cpp

3.12 Plyline Class Reference

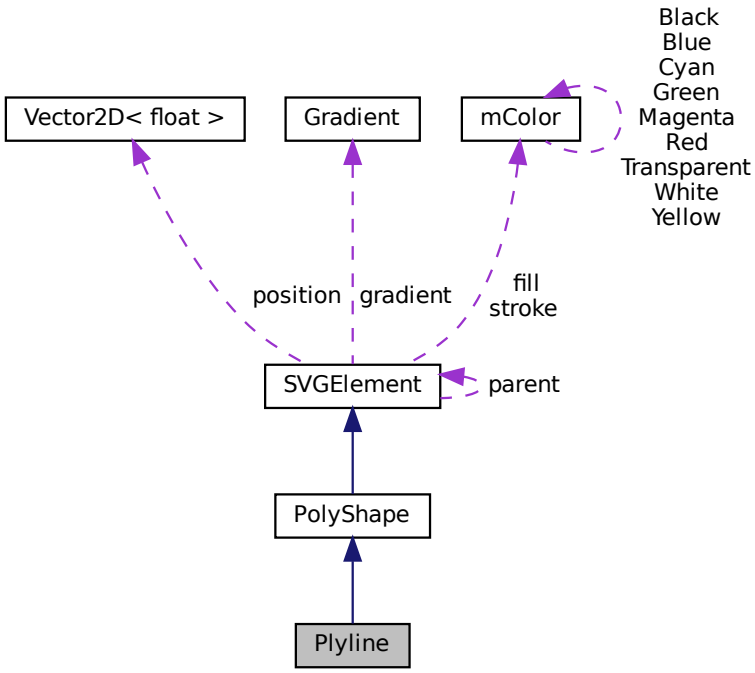
Represents a polyline in 2D space.

`#include <Polyline.hpp>`

Inheritance diagram for Plyline:



Collaboration diagram for Plyline:



Public Member Functions

- Plyline (const mColor &fill, const mColor &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 mColor & fill,
            const mColor & 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.

```
4     : 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 6 of file Polyline.cpp.

```
6 { return "Polyline"; }
```

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

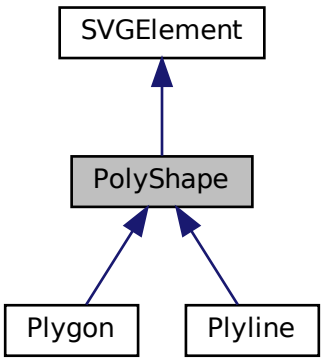
- src/graphics/Polyline.hpp
- src/graphics/Polyline.cpp

3.13 PolyShape Class Reference

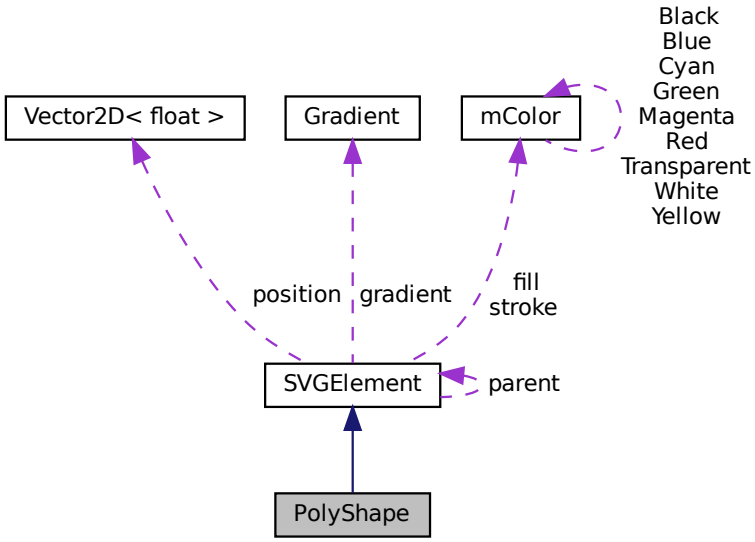
Abstract base class for polygon and polyline shapes in 2D space.

`#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 mColor &fill, const mColor &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 mColor & fill,
            const mColor & 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:

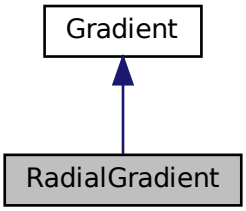
- src/graphics/PolyShape.hpp
- src/graphics/PolyShape.cpp

3.14 RadialGradient Class Reference

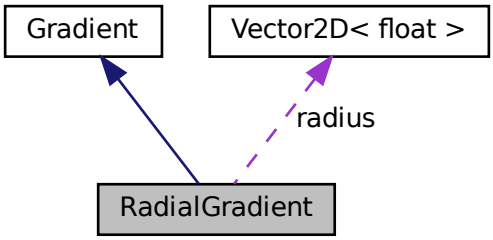
A class that represents a radial gradient.

`#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:

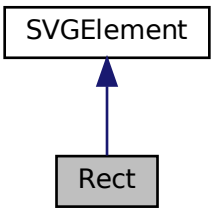
- src/graphics/RadialGradient.hpp
- src/graphics/RadialGradient.cpp

3.15 Rect Class Reference

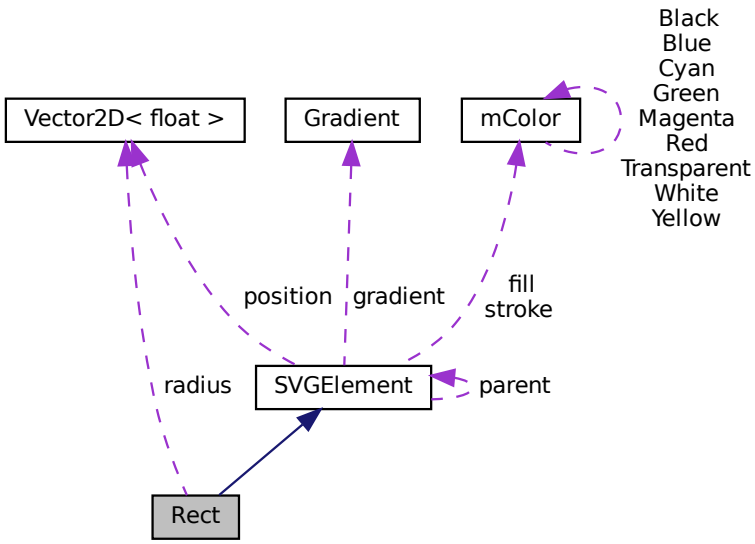
Represents a rectangle in 2D space.

`#include <Rect.hpp>`

Inheritance diagram for Rect:



Collaboration diagram for Rect:



Public Member Functions

- Rect (float width, float height, Vector2Df position, Vector2Df radius, const mColor &fill, const mColor &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 mColor & fill,
            const mColor & 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:

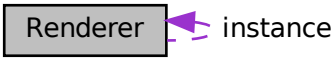
- src/graphics/Rect.hpp
- src/graphics/Rect.cpp

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 50 of file Renderer.cpp.

```
51                                                                  {
52     for (auto type : transform_order) {
53         if (type.find("translate") != std::string::npos) {
54             float trans_x = getTranslate(type).first,
```

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

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 673 of file Renderer.cpp.

```
675                                                 {
676      for (auto type : transform_order) {
677          if (type.find("translate") != std::string::npos) {
678              float trans_x = getTranslate(type).first,
679                    trans_y = getTranslate(type).second;
680              brush->TranslateTransform(trans_x, trans_y);
681          } else if (type.find("rotate") != std::string::npos) {
682              float degree = getRotate(type);
683              brush->RotateTranform(degree);
684          } else if (type.find("scale") != std::string::npos) {
685              if (type.find(",") != std::string::npos) {
686                  float scale_x = getScaleXY(type).first,
687                        scale_y = getScaleXY(type).second;
688                  brush->ScaleTransform(scale_x, scale_y);
689              } else {
690                  float scale = getScale(type);
691                  brush->ScaleTransform(scale, scale);
692              }
693          } else if (type.find("matrix") != std::string::npos) {
694              float a = 0, b = 0, c = 0, d = 0, e = 0, f = 0;
695              if (type.find(",") != std::string::npos) {
696                  type.erase(std::remove(type.begin(), type.end(), ','),
697                             type.end());
698              }
699              sscanf(type.c_str(), "matrix(%f %f %f %f %f %f)", &a, &b, &c, &d,
700                     &e, &f);
701              Gdiplus::Matrix matrix(a, b, c, d, e, f);
702              brush->SetTransform(&matrix);
703          }
704      }
705  }
```

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 707 of file Renderer.cpp.

```
709                                                 {
710     for (auto type : transform_order) {
711         if (type.find("translate") != std::string::npos) {
712             float trans_x = getTranslate(type).first,
713                   trans_y = getTranslate(type).second;
714             brush->TranslateTransform(trans_x, trans_y);
715         } else if (type.find("rotate") != std::string::npos) {
716             float degree = getRotate(type);
717             brush->RotateTransform(degree);
718         } else if (type.find("scale") != std::string::npos) {
719             if (type.find(",") != std::string::npos) {
720                 float scale_x = getScaleXY(type).first,
721                       scale_y = getScaleXY(type).second;
722                 brush->ScaleTransform(scale_x, scale_y);
723             } else {
724                 float scale = getScale(type);
725                 brush->ScaleTransform(scale, scale);
726             }
727         } else if (type.find("matrix") != std::string::npos) {
728             float a = 0, b = 0, c = 0, d = 0, e = 0, f = 0;
729             if (type.find(",") != std::string::npos) {
730                 type.erase(std::remove(type.begin(), type.end(), ','),
731                            type.end());
732             }
733             sscanf(type.c_str(), "matrix(%f %f %f %f %f %f)", &a, &b, &c, &d,
734                    &e, &f);
735             Gdiplus::Matrix matrix(a, b, c, d, e, f);
736             brush->SetTransform(&matrix);
737         }
738     }
739 }
```

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 73 of file Renderer.cpp.

```
73                                                                      {
74     for (auto shape : group->getElements()) {
75         Gdiplus::Matrix original;
76         graphics.GetTransform(&original);
77         applyTransform(shape->getTransforms(), graphics);
78         if (shape->getClass() == "Group") {
79             Group* group = dynamic_cast< Group* >(shape);
```

```
80              draw(graphics, group);
81          } else if (shape->getClass() == "Polyline") {
82              Plyline* polyline = dynamic_cast< Plyline* >(shape);
83              drawPolyline(graphics, polyline);
84          } else if (shape->getClass() == "Text") {
85              Text* text = dynamic_cast< Text* >(shape);
86              drawText(graphics, text);
87          } else if (shape->getClass() == "Rect") {
88              Rect* rectangle = dynamic_cast< Rect* >(shape);
89              drawRectangle(graphics, rectangle);
90          } else if (shape->getClass() == "Circle") {
91              Circle* circle = dynamic_cast< Circle* >(shape);
92              drawCircle(graphics, circle);
93          } else if (shape->getClass() == "Ellipse") {
94              Ell* ellipse = dynamic_cast< Ell* >(shape);
95              drawEllipse(graphics, ellipse);
96          } else if (shape->getClass() == "Line") {
97              Line* line = dynamic_cast< Line* >(shape);
98              drawLine(graphics, line);
99          } else if (shape->getClass() == "Polygon") {
100              Plygon* polygon = dynamic_cast< Plygon* >(shape);
101              drawPolygon(graphics, polygon);
102          } else if (shape->getClass() == "Path") {
103              Path* path = dynamic_cast< Path* >(shape);
104              drawPath(graphics, path);
105          }
106          graphics.SetTransform(&original);
107      }
108  }
```

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 165 of file Renderer.cpp.

```
165                                                                     {
166     mColor outline_color = circle->getOutlineColor();
167     Gdiplus::Pen circle_outline(
168         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
169                        outline_color.b),
170         circle->getOutlineThickness());
171     Vector2Df min_bound = circle->getMinBound();
172     Vector2Df max_bound = circle->getMaxBound();
173     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
174                          max_bound.y - min_bound.y);
175     Gdiplus::Brush* circle_fill = getBrush(circle, bound);
176     if (Gdiplus::PathGradientBrush* brush =
177             dynamic_cast< Gdiplus::PathGradientBrush* >(circle_fill)) {
178         mColor color = circle->getGradient()->getStops().back().getColor();
179         Gdiplus::SolidBrush corner_fill(
180             Gdiplus::Color(color.a, color.r, color.g, color.b));
181         graphics.FillEllipse(
182             &corner_fill, circle->getPosition().x - circle->getRadius().x,
183             circle->getPosition().y - circle->getRadius().y,
184             circle->getRadius().x * 2, circle->getRadius().y * 2);
185     }
186     graphics.FillEllipse(circle_fill,
187                          circle->getPosition().x - circle->getRadius().x,
188                          circle->getPosition().y - circle->getRadius().y,
189                          circle->getRadius().x * 2, circle->getRadius().y * 2);
190     graphics.DrawEllipse(&circle_outline,
191                          circle->getPosition().x - circle->getRadius().x,
```

```
192                         circle->getPosition().y - circle->getRadius().y,
193                         circle->getRadius().x * 2, circle->getRadius().x * 2);
194     delete circle_fill;
195 }
```

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 197 of file Renderer.cpp.

```
197                                                                         {
198     mColor outline_color = ellipse->getOutlineColor();
199     Gdiplus::Pen ellipse_outline(
200         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
201                        outline_color.b),
202         ellipse->getOutlineThickness());
203     Vector2Df min_bound = ellipse->getMinBound();
204     Vector2Df max_bound = ellipse->getMaxBound();
205     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
206                          max_bound.y - min_bound.y);
207     Gdiplus::Brush* ellipse_fill = getBrush(ellipse, bound);
208     if (Gdiplus::PathGradientBrush* brush =
209             dynamic_cast< Gdiplus::PathGradientBrush* >(ellipse_fill)) {
210         mColor color = ellipse->getGradient()->getStops().back().getColor();
211         Gdiplus::SolidBrush corner_fill(
212             Gdiplus::Color(color.a, color.r, color.g, color.b));
213         graphics.FillEllipse(
214             &corner_fill, ellipse->getPosition().x - ellipse->getRadius().x,
215             ellipse->getPosition().y - ellipse->getRadius().y,
216             ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
217     }
218     graphics.FillEllipse(
219         ellipse_fill, ellipse->getPosition().x - ellipse->getRadius().x,
220         ellipse->getPosition().y - ellipse->getRadius().y,
221         ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
222     graphics.DrawEllipse(
223         &ellipse_outline, ellipse->getPosition().x - ellipse->getRadius().x,
224         ellipse->getPosition().y - ellipse->getRadius().y,
225         ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
226     delete ellipse_fill;
227 }
```

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 110 of file Renderer.cpp.

```
110                                                                 {
111     mColor color = line->getOutlineColor();
112     Gdiplus::Pen linePen(Gdiplus::Color(color.a, color.r, color.g, color.b),
113                          line->getOutlineThickness());
114     Gdiplus::PointF startPoint(line->getPosition().x, line->getPosition().y);
115     Gdiplus::PointF endPoint(line->getDirection().x, line->getDirection().y);
116     graphics.DrawLine(&linePen, startPoint, endPoint);
117 }
```

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 365 of file Renderer.cpp.

```
365                                                                 {
366     mColor outline_color = path->getOutlineColor();
367     Gdiplus::Pen path_outline(Gdiplus::Color(outline_color.a, outline_color.r,
368                                              outline_color.g, outline_color.b),
369                               path->getOutlineThickness());
370
371     Gdiplus::FillMode fill_mode;
372     if (path->getFillRule() == "evenodd") {
373         fill_mode = Gdiplus::FillModeAlternate;
374     } else if (path->getFillRule() == "nonzero") {
375         fill_mode = Gdiplus::FillModeWinding;
376     }
377     Gdiplus::GraphicsPath gdi_path(fill_mode);
378
379     const std::vector< PathPoint >& points = path->getPoints();
380     int n = points.size();
381     Vector2Df first_point{0, 0}, cur_point{0, 0};
382
383     for (int i = 0; i < n; ++i) {
384         if (points[i].tc == 'm') {
385             first_point = points[i].point;
386             gdi_path.StartFigure();
387             cur_point = first_point;
388         } else if (points[i].tc == 'l' || points[i].tc == 'h' ||
389                    points[i].tc == 'v') {
390             gdi_path.AddLine(cur_point.x, cur_point.y, points[i].point.x,
391                              points[i].point.y);
392             cur_point = points[i].point;
393         } else if (points[i].tc == 'c') {
394             if (i + 2 < n) {
395                 Vector2Df control_point1 = points[i].point;
396                 Vector2Df control_point2 = points[i + 1].point;
397                 Vector2Df control_point3 = points[i + 2].point;
398                 gdi_path.AddBezier(cur_point.x, cur_point.y, control_point1.x,
399                                    control_point1.y, control_point2.x,
400                                    control_point2.y, control_point3.x,
401                                    control_point3.y);
```

```
402                 i += 2;
403                 cur_point = control_point3;
404             }
405         } else if (points[i].tc == 'z') {
406             gdi_path.CloseFigure();
407             cur_point = first_point;
408         } else if (points[i].tc == 's') {
409             if (i + 1 < n) {
410                 Vector2Df auto_control_point;
411                 if (i > 0 &&
412                     (points[i - 1].tc == 'c' || points[i - 1].tc == 's')) {
413                     auto_control_point.x =
414                         cur_point.x * 2 - points[i - 2].point.x;
415                     auto_control_point.y =
416                         cur_point.y * 2 - points[i - 2].point.y;
417                 } else {
418                     auto_control_point = cur_point;
419                 }
420                 Vector2Df control_point2 = points[i].point;
421                 Vector2Df control_point3 = points[i + 1].point;
422                 gdi_path.AddBezier(cur_point.x, cur_point.y,
423                                    auto_control_point.x, auto_control_point.y,
424                                    control_point2.x, control_point2.y,
425                                    control_point3.x, control_point3.y);
426                 i += 1;
427                 cur_point = control_point3;
428             }
429         } else if (points[i].tc == 'q') {
430             if (i + 1 < n) {
431                 Vector2Df control_point = points[i].point;
432                 Vector2Df end_point = points[i + 1].point;
433 
434                 Gdiplus::PointF q_points[3];
435                 q_points[0] = Gdiplus::PointF{cur_point.x, cur_point.y};
436                 q_points[1] = Gdiplus::PointF{control_point.x, control_point.y};
437                 q_points[2] = Gdiplus::PointF{end_point.x, end_point.y};
438                 gdi_path.AddCurve(q_points, 3);
439                 cur_point = points[i + 1].point;
440                 i += 1;
441             }
442         } else if (points[i].tc == 't') {
443             Vector2Df auto_control_point;
444             if (i > 0 && (points[i - 1].tc == 'q' || points[i - 1].tc == 't')) {
445                 auto_control_point.x = cur_point.x * 2 - points[i - 2].point.x;
446                 auto_control_point.y = cur_point.y * 2 - points[i - 2].point.y;
447             } else {
448                 auto_control_point = cur_point;
449             }
450             Vector2Df end_point = points[i].point;
451             Gdiplus::PointF t_points[3];
452             t_points[0] = Gdiplus::PointF{cur_point.x, cur_point.y};
453             t_points[1] =
454                 Gdiplus::PointF{auto_control_point.x, auto_control_point.y};
455             t_points[2] = Gdiplus::PointF{end_point.x, end_point.y};
456             gdi_path.AddCurve(t_points, 3);
457             cur_point = points[i].point;
458         } else if (points[i].tc == 'a') {
459             float rx = points[i].radius.x;
460             float ry = points[i].radius.y;
461             if (rx == 0 || ry == 0) {
462                 gdi_path.AddLine(cur_point.x, cur_point.y, points[i].point.x,
463                                  points[i].point.y);
464                 cur_point = points[i].point;
465                 continue;
466             }
467             if (rx < 0) {
468                 rx = std::fabs(rx);
469             }
470             if (ry < 0) {
471                 ry = std::fabs(ry);
472             }
473 
474             float x_axis_rotation = points[i].x_axis_rotation;
475             bool large_arc_flag = points[i].large_arc_flag;
476             bool sweep_flag = points[i].sweep_flag;
477             Vector2Df end_point{points[i].point.x, points[i].point.y};
478 
479             float angle = x_axis_rotation * acos(-1) / 180.0;
480             float cosAngle = cos(angle);
481             float sinAngle = sin(angle);
482 
483             Vector2Df point1;
484             float X = (cur_point.x - end_point.x) / 2.0;
485             float Y = (cur_point.y - end_point.y) / 2.0;
486             point1.x = (cosAngle * cosAngle + sinAngle * sinAngle) * X;
487             point1.y = (cosAngle * cosAngle + sinAngle * sinAngle) * Y;
488 
```

```
489             float radii_check = (point1.x * point1.x) / (rx * rx) +
490                                 (point1.y * point1.y) / (ry * ry);
491             if (radii_check > 1.0) {
492                 rx = std::sqrt(radii_check) * rx;
493                 ry = std::sqrt(radii_check) * ry;
494             }
495 
496             float sign = (large_arc_flag == sweep_flag ? -1.0 : 1.0);
497             Vector2Df point2;
498             float numo = (rx * rx) * (ry * ry) -
499                          (rx * rx) * (point1.y * point1.y) -
500                          (ry * ry) * (point1.x * point1.x);
501             float deno = (rx * rx) * (point1.y * point1.y) +
502                          (ry * ry) * (point1.x * point1.x);
503 
504             if (numo < 0) {
505                 numo = std::fabs(numo);
506             }
507 
508             point2.x = sign * std::sqrt(numo / deno) * ((rx * point1.y) / ry);
509             point2.y = sign * std::sqrt(numo / deno) * ((-ry * point1.x) / rx);
510 
511             Vector2Df center;
512             X = (cur_point.x + end_point.x) / 2.0;
513             Y = (cur_point.y + end_point.y) / 2.0;
514             center.x =
515                 (cosAngle * cosAngle + sinAngle * sinAngle) * point2.x + X;
516             center.y =
517                 (cosAngle * cosAngle + sinAngle * sinAngle) * point2.y + Y;
518 
519             float start_angle =
520                 atan2((point1.y - point2.y) / ry, (point1.x - point2.x) / rx);
521             float end_angle =
522                 atan2((-point1.y - point2.y) / ry, (-point1.x - point2.x) / rx);
523 
524             float delta_angle = end_angle - start_angle;
525 
526             if (sweep_flag && delta_angle < 0) {
527                 delta_angle += 2.0 * acos(-1);
528             } else if (!sweep_flag && delta_angle > 0) {
529                 delta_angle -= 2.0 * acos(-1);
530             }
531 
532             float start_angle_degree =
533                 std::fmod((start_angle * 180.0) / acos(-1), 360);
534             float delta_angle_degree =
535                 std::fmod((delta_angle * 180.0) / acos(-1), 360);
536 
537             gdi_path.AddArc(center.x - rx, center.y - ry, 2.0 * rx, 2.0 * ry,
538                             start_angle_degree, delta_angle_degree);
539 
540             cur_point = end_point;
541         }
542     }
543 
544     Gdiplus::RectF bound;
545     gdi_path.GetBounds(&bound);
546     Gdiplus::Brush* path_fill = getBrush(path, bound);
547     Gdiplus::Region region(&gdi_path);
548     if (Gdiplus::PathGradientBrush* brush =
549             dynamic_cast< Gdiplus::PathGradientBrush* >(path_fill)) {
550         mColor color = path->getGradient()->getStops().back().getColor();
551         Gdiplus::SolidBrush corner_fill(
552             Gdiplus::Color(color.a, color.r, color.g, color.b));
553         if (path->getGradient()->getUnits() == "userSpaceOnUse") {
554             float cx = path->getGradient()->getPoints().first.x;
555             float cy = path->getGradient()->getPoints().first.y;
556             float r = dynamic_cast< RadialGradient* >(path->getGradient())
557                           ->getRadius()
558                           .x;
559             Gdiplus::GraphicsPath fill_path(fill_mode);
560             fill_path.AddEllipse(cx - r, cy - r, 2 * r, 2 * r);
561             for (auto type : path->getGradient()->getTransforms()) {
562                 if (type.find("matrix") != std::string::npos) {
563                     float a = 0, b = 0, c = 0, d = 0, e = 0, f = 0;
564                     if (type.find(",") != std::string::npos) {
565                         type.erase(std::remove(type.begin(), type.end(), ','),
566                                    type.end());
567                     }
568                     sscanf(type.c_str(), "matrix(%f %f %f %f %f %f)", &a, &b,
569                            &c, &d, &e, &f);
570                     Gdiplus::Matrix matrix(a, b, c, d, e, f);
571                     fill_path.Transform(&matrix);
572                 }
573             }
574             region.Exclude(&fill_path);
575         }
```

```
576         graphics.FillRegion(&corner_fill, &region);
577     }
578     graphics.FillPath(path_fill, &gdi_path);
579     graphics.DrawPath(&path_outline, &gdi_path);
580     delete path_fill;
581 }
```

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 229 of file Renderer.cpp.

```
229                                                                                 {
230     mColor outline_color = polygon->getOutlineColor();
231     Gdiplus::Pen polygon_outline(
232         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
233                        outline_color.b),
234         polygon->getOutlineThickness());
235
236     Gdiplus::PointF* points = new Gdiplus::PointF[polygon->getPoints().size()];
237     int idx = 0;
238     const std::vector< Vector2Df >& vertices = polygon->getPoints();
239     for (const Vector2Df vertex : vertices) {
240         points[idx++] = Gdiplus::PointF(vertex.x, vertex.y);
241     }
242
243     Gdiplus::FillMode fill_mode;
244     if (polygon->getFillRule() == "evenodd") {
245         fill_mode = Gdiplus::FillModeAlternate;
246     } else if (polygon->getFillRule() == "nonzero") {
247         fill_mode = Gdiplus::FillModeWinding;
248     }
249     Vector2Df min_bound = polygon->getMinBound();
250     Vector2Df max_bound = polygon->getMaxBound();
251     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
252                          max_bound.y - min_bound.y);
253     Gdiplus::Brush* polygon_fill = getBrush(polygon, bound);
254     if (Gdiplus::PathGradientBrush* brush =
255             dynamic_cast< Gdiplus::PathGradientBrush* >(polygon_fill)) {
256         mColor color = polygon->getGradient()->getStops().back().getColor();
257         Gdiplus::SolidBrush corner_fill(
258             Gdiplus::Color(color.a, color.r, color.g, color.b));
259         graphics.FillPolygon(&corner_fill, points, idx, fill_mode);
260     }
261     graphics.FillPolygon(polygon_fill, points, idx, fill_mode);
262     graphics.DrawPolygon(&polygon_outline, points, idx);
263     delete[] points;
264     delete polygon_fill;
265 }
```

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 322 of file Renderer.cpp.

```
323                                                 {
324     mColor outline_color = polyline->getOutlineColor();
325     Gdiplus::Pen polyline_outline(
326         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
327                        outline_color.b),
328         polyline->getOutlineThickness());
329
330     Gdiplus::FillMode fill_mode;
331     if (polyline->getFillRule() == "evenodd") {
332         fill_mode = Gdiplus::FillModeAlternate;
333     } else if (polyline->getFillRule() == "nonzero") {
334         fill_mode = Gdiplus::FillModeWinding;
335     }
336     Gdiplus::GraphicsPath path(fill_mode);
337     const std::vector< Vector2Df >& points = polyline->getPoints();
338     if (points.size() < 2) {
339         return;
340     }
341
342     path.StartFigure();
343     path.AddLine(points[0].x, points[0].y, points[1].x, points[1].y);
344     for (size_t i = 2; i < points.size(); ++i) {
345         path.AddLine(points[i - 1].x, points[i - 1].y, points[i].x,
346                      points[i].y);
347     }
348     Vector2Df min_bound = polyline->getMinBound();
349     Vector2Df max_bound = polyline->getMaxBound();
350     Gdiplus::RectF bound(min_bound.x, min_bound.y, max_bound.x - min_bound.x,
351                          max_bound.y - min_bound.y);
352     Gdiplus::Brush* polyline_fill = getBrush(polyline, bound);
353     if (Gdiplus::PathGradientBrush* brush =
354             dynamic_cast< Gdiplus::PathGradientBrush* >(polyline_fill)) {
355         mColor color = polyline->getGradient()->getStops().back().getColor();
356         Gdiplus::SolidBrush corner_fill(
357             Gdiplus::Color(color.a, color.r, color.g, color.b));
358         graphics.FillPath(&corner_fill, &path);
359     }
360     graphics.FillPath(polyline_fill, &path);
361     graphics.DrawPath(&polyline_outline, &path);
362     delete polyline_fill;
363 }
```

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 119 of file Renderer.cpp.

```
120                                                 {
121     float x = rectangle->getPosition().x;
122     float y = rectangle->getPosition().y;
123     float width = rectangle->getWidth();
```

```
124     float height = rectangle->getHeight();
125     mColor outline_color = rectangle->getOutlineColor();
126     Gdiplus::Pen rect_outline(Gdiplus::Color(outline_color.a, outline_color.r,
127                                              outline_color.g, outline_color.b),
128                               rectangle->getOutlineThickness());
129     Gdiplus::RectF bound(x, y, width, height);
130     Gdiplus::Brush* rect_fill = getBrush(rectangle, bound);
131     if (rectangle->getRadius().x != 0 || rectangle->getRadius().y != 0) {
132         float dx = rectangle->getRadius().x * 2;
133         float dy = rectangle->getRadius().y * 2;
134         Gdiplus::GraphicsPath path;
135         path.AddArc(x, y, dx, dy, 180, 90);
136         path.AddArc(x + width - dx, y, dx, dy, 270, 90);
137         path.AddArc(x + width - dx, y + height - dy, dx, dy, 0, 90);
138         path.AddArc(x, y + height - dy, dx, dy, 90, 90);
139         path.CloseFigure();
140         if (Gdiplus::PathGradientBrush* brush =
141                 dynamic_cast< Gdiplus::PathGradientBrush* >(rect_fill)) {
142             mColor color =
143                 rectangle->getGradient()->getStops().back().getColor();
144             Gdiplus::SolidBrush corner_fill(
145                 Gdiplus::Color(color.a, color.r, color.g, color.b));
146             graphics.FillPath(&corner_fill, &path);
147         }
148         graphics.FillPath(rect_fill, &path);
149         graphics.DrawPath(&rect_outline, &path);
150     } else {
151         if (Gdiplus::PathGradientBrush* brush =
152                 dynamic_cast< Gdiplus::PathGradientBrush* >(rect_fill)) {
153             mColor color =
154                 rectangle->getGradient()->getStops().back().getColor();
155             Gdiplus::SolidBrush corner_fill(
156                 Gdiplus::Color(color.a, color.r, color.g, color.b));
157             graphics.FillRectangle(&corner_fill, x, y, width, height);
158         }
159         graphics.FillRectangle(rect_fill, x, y, width, height);
160         graphics.DrawRectangle(&rect_outline, x, y, width, height);
161     }
162     delete rect_fill;
163 }
```

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 267 of file Renderer.cpp.

```
267                                                                     {
268     mColor outline_color = text->getOutlineColor();
269     graphics.SetTextRenderingHint(Gdiplus::TextRenderingHintAntiAliasGridFit);
270
271     Gdiplus::Pen text_outline(Gdiplus::Color(outline_color.a, outline_color.r,
272                                              outline_color.g, outline_color.b),
273                               text->getOutlineThickness());
274
275     Gdiplus::FontFamily font_family(L"Times New Roman");
276
277     Gdiplus::PointF position(text->getPosition().x, text->getPosition().y);
278     Gdiplus::GraphicsPath path;
279
280     std::wstring_convert< std::codecvt_utf8_utf16< wchar_t > > converter;
281     std::wstring wide_content = converter.from_bytes(text->getContent());
282     Gdiplus::StringFormat string_format;
```

```
283     if (text->getAnchor() == "middle") {
284         string_format.SetAlignment(Gdiplus::StringAlignmentCenter);
285         position.X += 7;
286     } else if (text->getAnchor() == "end") {
287         string_format.SetAlignment(Gdiplus::StringAlignmentFar);
288         position.X += 14;
289     } else {
290         string_format.SetAlignment(Gdiplus::StringAlignmentNear);
291     }
292     Gdiplus::FontStyle font_style = Gdiplus::FontStyleRegular;
293     if (text->getFontStyle() == "italic" || text->getFontStyle() == "oblique") {
294         font_style = Gdiplus::FontStyleItalic;
295         position.Y -= 1;
296     }
297 
298     path.AddString(wide_content.c_str(), wide_content.size(), &font_family,
299                    font_style, text->getFontSize(), position, &string_format);
300     Gdiplus::RectF bound;
301     path.GetBounds(&bound);
302     Gdiplus::Brush* text_fill = getBrush(text, bound);
303     if (Gdiplus::PathGradientBrush* brush =
304             dynamic_cast< Gdiplus::PathGradientBrush* >(text_fill)) {
305         mColor color = text->getGradient()->getStops().back().getColor();
306         Gdiplus::SolidBrush corner_fill(
307             Gdiplus::Color(color.a, color.r, color.g, color.b));
308         graphics.FillPath(&corner_fill, &path);
309     }
310     graphics.FillPath(text_fill, &path);
311     if (text->getOutlineColor().a != 0 &&
312         text->getOutlineColor().a == text->getFillColor().a) {
313         text_outline.SetColor(Gdiplus::Color(255, 255, 255, 255));
314         graphics.DrawPath(&text_outline, &path);
315         text_outline.SetColor(Gdiplus::Color(outline_color.a, outline_color.r,
316                                              outline_color.g, outline_color.b));
317     }
318     graphics.DrawPath(&text_outline, &path);
319     delete text_fill;
320 }
```

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 583 of file Renderer.cpp.

```
584                                                                 {
585     Gradient* gradient = shape->getGradient();
586     if (gradient != NULL) {
587         std::pair< Vector2Df, Vector2Df > points = gradient->getPoints();
588         std::vector< Stop > stops = gradient->getStops();
589         int stop_size = stops.size() + 2;
590         Gdiplus::Color* colors = new Gdiplus::Color[stop_size];
591         float* offsets = new float[stop_size];
592         if (gradient->getClass() == "LinearGradient") {
593             if (gradient->getUnits() == "objectBoundingBox") {
594                 points.first.x = bound.X;
```

```
595                 points.first.y = bound.Y;
596                 points.second.x = bound.X + bound.Width;
597                 points.second.y = bound.Y + bound.Height;
598             }
599             offsets[0] = 0;
600             offsets[stop_size - 1] = 1;
601             colors[0] =
602                 Gdiplus::Color(stops[0].getColor().a, stops[0].getColor().r,
603                                stops[0].getColor().g, stops[0].getColor().b);
604             colors[stop_size - 1] =
605                 Gdiplus::Color(stops[stop_size - 3].getColor().a,
606                                stops[stop_size - 3].getColor().r,
607                                stops[stop_size - 3].getColor().g,
608                                stops[stop_size - 3].getColor().b);
609             for (size_t i = 1; i < stop_size - 1; ++i) {
610                 colors[i] = Gdiplus::Color(
611                     stops[i - 1].getColor().a, stops[i - 1].getColor().r,
612                     stops[i - 1].getColor().g, stops[i - 1].getColor().b);
613                 offsets[i] = stops[i - 1].getOffset();
614             }
615             Gdiplus::LinearGradientBrush* fill =
616                 new Gdiplus::LinearGradientBrush(
617                     Gdiplus::PointF(points.first.x, points.first.y),
618                     Gdiplus::PointF(points.second.x, points.second.y),
619                     colors[0], colors[stop_size - 1]);
620             fill->SetWrapMode(Gdiplus::WrapModeTileFlipX);
621             fill->SetInterpolationColors(colors, offsets, stop_size);
622             applyTransformsOnBrush(gradient->getTransforms(), fill);
623             delete[] colors;
624             delete[] offsets;
625             return fill;
626         } else if (gradient->getClass() == "RadialGradient") {
627             RadialGradient* radial_gradient =
628                 dynamic_cast< RadialGradient* >(gradient);
629             Vector2Df radius = radial_gradient->getRadius();
630             if (gradient->getUnits() == "userSpaceOnUse") {
631                 bound.X = points.first.x - radius.x;
632                 bound.Y = points.first.y - radius.x;
633                 bound.Width = radius.x * 2;
634                 bound.Height = radius.x * 2;
635             }
636             Gdiplus::GraphicsPath path;
637             path.AddEllipse(bound);
638             Gdiplus::PathGradientBrush* fill =
639                 new Gdiplus::PathGradientBrush(&path);
640             offsets[0] = 0;
641             offsets[stop_size - 1] = 1;
642             colors[0] = Gdiplus::Color(stops[stop_size - 3].getColor().a,
643                                        stops[stop_size - 3].getColor().r,
644                                        stops[stop_size - 3].getColor().g,
645                                        stops[stop_size - 3].getColor().b);
646             colors[stop_size - 1] =
647                 Gdiplus::Color(stops[0].getColor().a, stops[0].getColor().r,
648                                stops[0].getColor().g, stops[0].getColor().b);
649
650             for (size_t i = 1; i < stop_size - 1; ++i) {
651                 colors[i] =
652                     Gdiplus::Color(stops[stop_size - 2 - i].getColor().a,
653                                    stops[stop_size - 2 - i].getColor().r,
654                                    stops[stop_size - 2 - i].getColor().g,
655                                    stops[stop_size - 2 - i].getColor().b);
656                 offsets[i] = 1 - stops[stop_size - 2 - i].getOffset();
657             }
658             fill->SetInterpolationColors(colors, offsets, stop_size);
659             applyTransformsOnBrush(gradient->getTransforms(), fill);
660             delete[] colors;
661             delete[] offsets;
662             return fill;
663         }
664     } else {
665         mColor color = shape->getFillColor();
666         Gdiplus::SolidBrush* fill = new Gdiplus::SolidBrush(
667             Gdiplus::Color(color.a, color.r, color.g, color.b));
668         return fill;
669     }
670     return nullptr;
671 }
```

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:

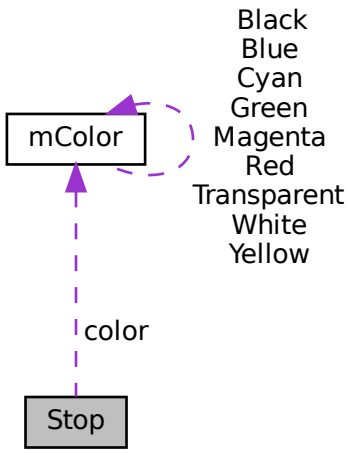
- src/Renderer.hpp
- src/Renderer.cpp

3.17 Stop Class Reference

A class that represents a stop.

`#include <Stop.hpp>`

Collaboration diagram for Stop:



Public Member Functions

- Stop (const mColor &color, float offset)

  Constructs a Stop object.
- mColor getColor () const

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

  Gets the offset of the stop.

Private Attributes

- mColor 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 mColor & 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.

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

3.17.3 Member Function Documentation

3.17.3.1 getColor()

```
mColor Stop::getColor ( ) const
```

Gets the color of the stop.

Returns

The color of the stop.

Definition at line 5 of file Stop.cpp.

```
5 { 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 7 of file Stop.cpp.

```
7 { return offset; }
```

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

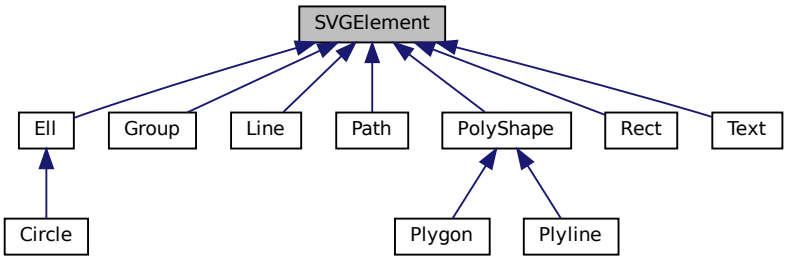
- src/graphics/Stop.hpp
- src/graphics/Stop.cpp

3.18 SVGElement Class Reference

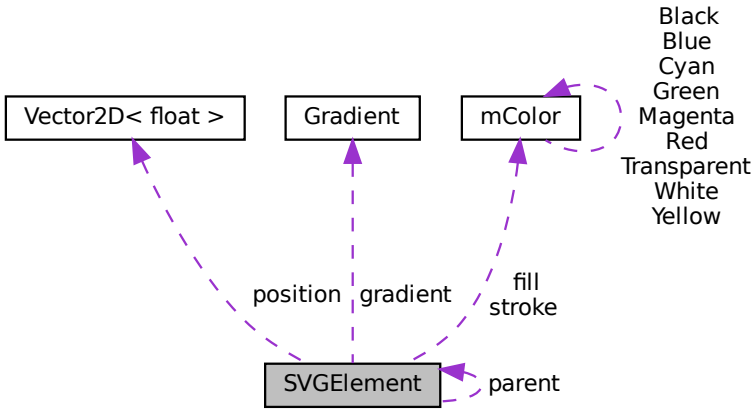
Represents an element in an SVG file.

```
#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 mColor &color)

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

  Gets the fill color of the shape.
- const mColor & 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 mColor &fill, const mColor &stroke, float stroke_width)

  Constructs a Shape object.
- SVGElement (const mColor &fill, const mColor &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

- mColor fill

  Fill color.
- mColor 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(mColor::Black), stroke(mColor::Transparent), stroke_width(1),
7       gradient(NULL) {}
```

3.18.2.2 SVGElement() [2/3]

```
SVGElement::SVGElement (
            const mColor & fill,
            const mColor & 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 mColor & fill,
            const mColor & 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 mColor & 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 mColor & 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 mColor & 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 mColor & 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:

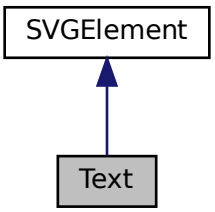
- src/graphics/SVGElement.hpp
- src/graphics/SVGElement.cpp

3.19 Text Class Reference

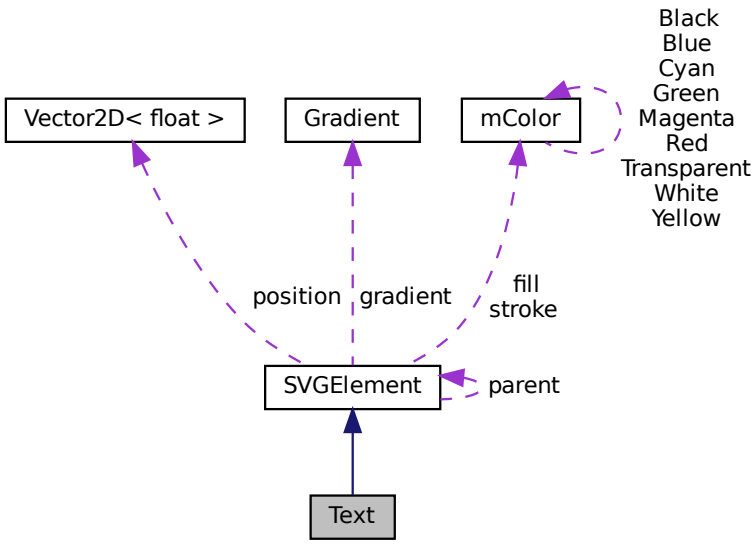
Represents text in 2D space.

`#include <Text.hpp>`

Inheritance diagram for Text:



Collaboration diagram for Text:



Public Member Functions

- Text (Vector2Df pos, std::string text, float font_size, const mColor &fill, const mColor &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 mColor & fill,
            const mColor & 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:

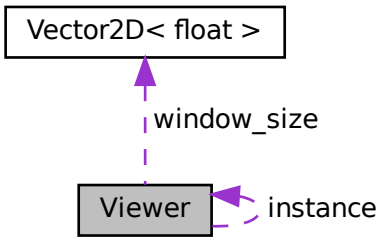
- src/graphics/Vector2D.hpp

3.21 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.

3.21.1 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.

3.21.2 Member Function Documentation

3.21.2.1 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 }
```

3.21.2.2 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 }
```

3.21.2.3 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 }
```

3.21.2.4 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); }
```

3.21.2.5 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  }
```

3.21.2.6 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 }
```

3.21.2.7 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 }
```

3.21.2.8 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 }
```

3.21.3 Member Data Documentation

3.21.3.1 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