svg-reader

0.2

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

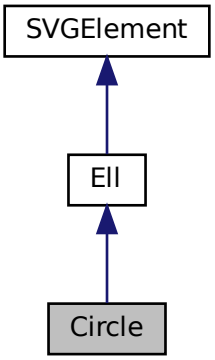
Class Documentation

3.1 Circle Class Reference

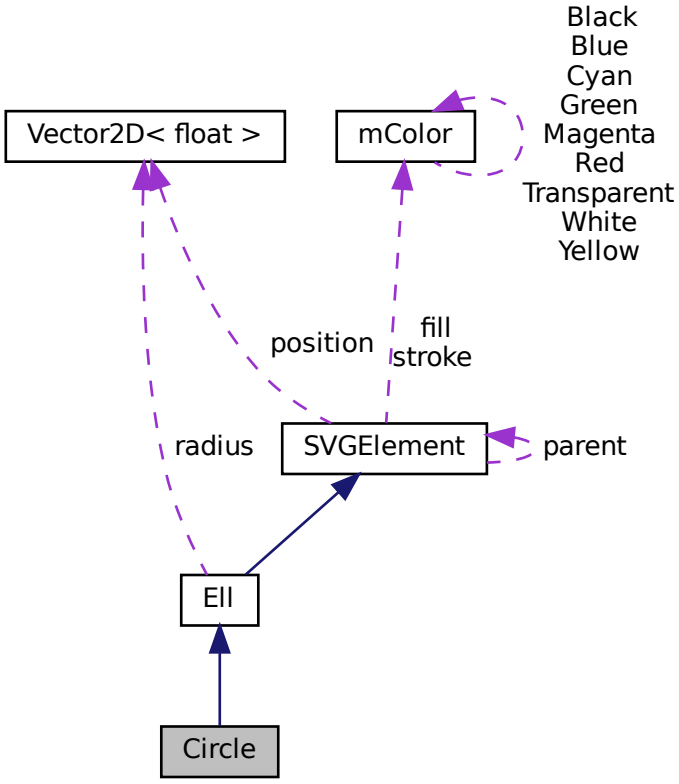
Represents a circle in 2D space.

`#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:

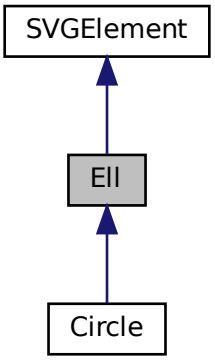
- src/graphics/Circle.hpp
- src/graphics/Circle.cpp

3.2 Ell Class Reference

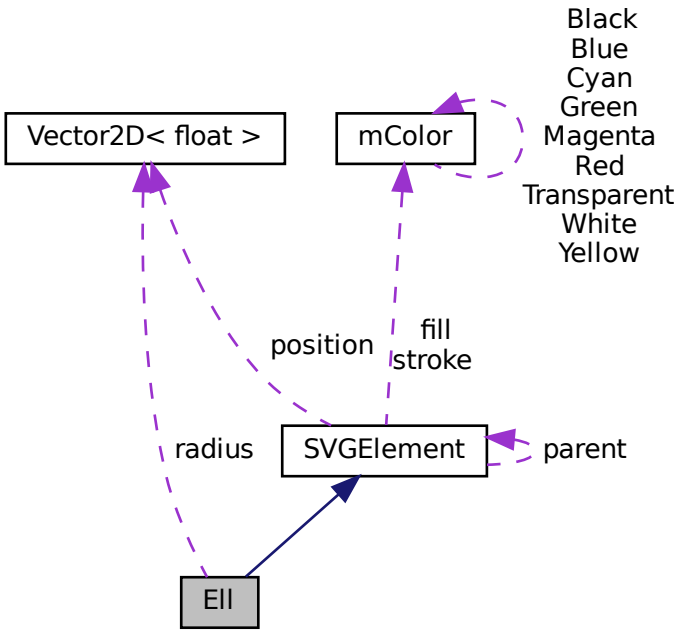
Represents an ellipse in 2D space.

`#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.
- 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     : radius(radius) {
8     setPosition(center);
9     setFillColor(fill);
10     setOutlineColor(stroke);
11     setOutlineThickness(stroke_thickness);
12 }
```

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 14 of file Ellipse.cpp.

```
14 { return "Ellipse"; }
```

3.2.3.2 getRadius()

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

Gets the radius of the ellipse.

Returns

The radius of the ellipse.

Definition at line 18 of file Ellipse.cpp.

```
18 { return radius; }
```

3.2.3.3 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 20 of file Ellipse.cpp.

```
20                              {
21     SVGElement::printData();
22     std::cout « "Radius: " « getRadius().x « " " « getRadius().y
23               « std::endl;
24 }
```

3.2.3.4 setRadius()

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

Sets the radius of the ellipse.

Parameters

radius	The new radius of the ellipse.

Definition at line 16 of file Ellipse.cpp.

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

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

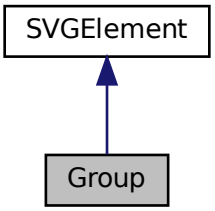
- src/graphics/Ellipse.hpp
- src/graphics/Ellipse.cpp

3.3 Group Class Reference

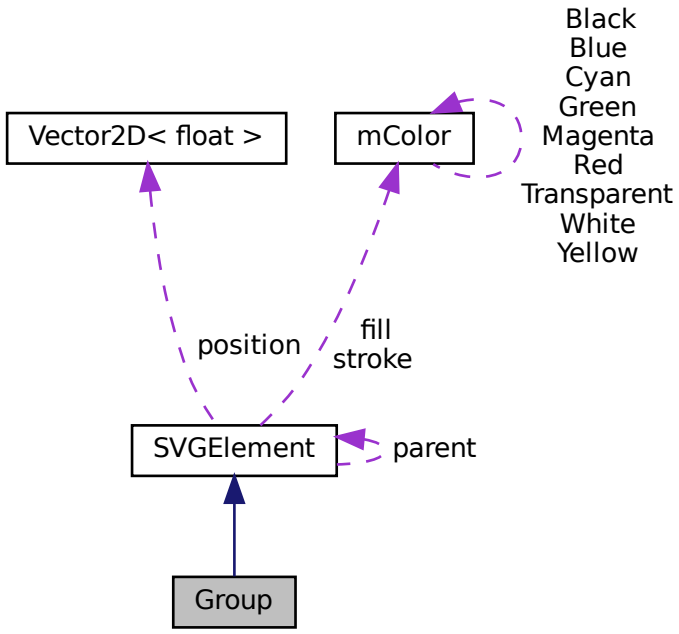
A composite class that contains a vector of shape pointers (polymorphic).

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

Inheritance diagram for Group:



Collaboration diagram for Group:



Public Member Functions

- **Group** (Attributes attributes)
- 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

3.3.1 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.

3.3.2 Member Function Documentation

3.3.2.1 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 }
```

3.3.2.2 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; }
```

3.3.2.3 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"; }
```

3.3.2.4 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; }
```

3.3.2.5 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:

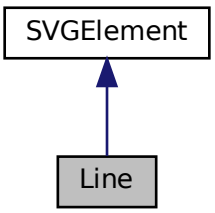
- src/graphics/Group.hpp
- src/graphics/Group.cpp

3.4 Line Class Reference

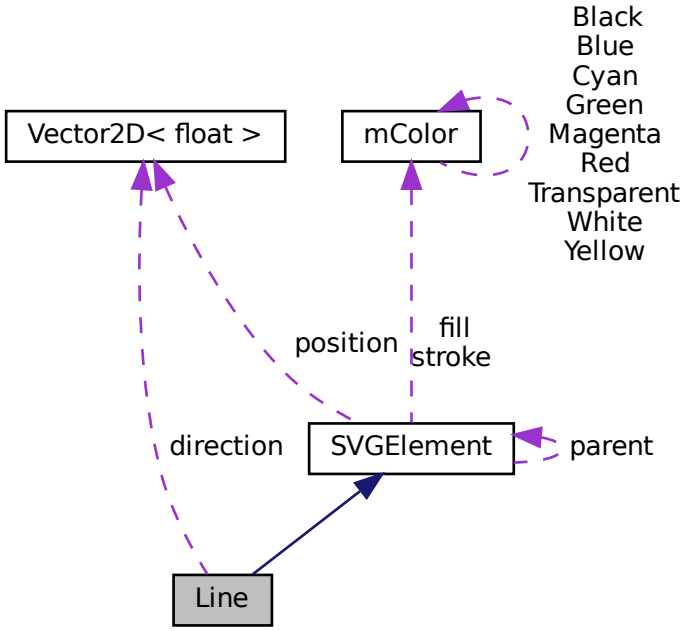
Represents a line in 2D space.

```
#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

3.4.1 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.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 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     : direction(point2) {
8     setPosition(point1);
9     setOutlineThickness(stroke_width);
10     setOutlineColor(stroke);
11 }
```

3.4.3 Member Function Documentation

3.4.3.1 getClass()

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

Gets the type of the shape.

Returns

The string "Line".

Implements SVGElement.

Definition at line 13 of file Line.cpp.

```
13 { return "Line"; }
```

3.4.3.2 getDirection()

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

Gets the direction of the line.

Returns

The direction of the line.

Definition at line 19 of file Line.cpp.

```
19 { return direction; }
```

3.4.3.3 getLength()

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

Gets the length of the line.

Returns

The length of the line.

Definition at line 21 of file Line.cpp.

```
21                              {
22     return std::sqrt(std::pow(direction.x, 2) + std::pow(direction.y, 2));
23 }
```

3.4.3.4 setDirection()

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

Sets the direction of the line.

Parameters

direction	The new direction of the line.

Definition at line 15 of file Line.cpp.

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

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

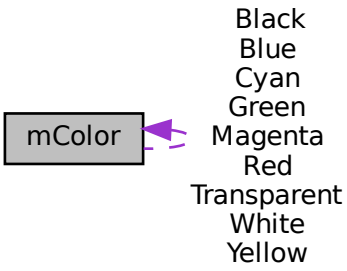
- src/graphics/Line.hpp
- src/graphics/Line.cpp

3.5 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.

3.5.1 Detailed Description

Utility class for manipulating RGBA mColors.

Definition at line 11 of file Color.hpp.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 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) {}
```

3.5.2.2 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 }
```

3.5.2.3 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) {}
```

3.5.3 Friends And Related Function Documentation

3.5.3.1 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:

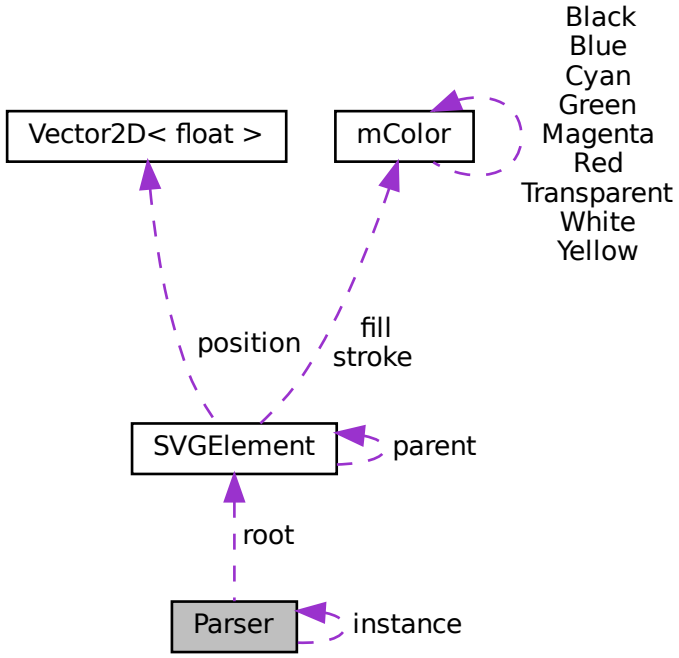
- src/graphics/Color.hpp
- src/graphics/Color.cpp

3.6 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.
- Group * getRoot ()

  Gets the root of the SVG file.
- void printShapesData ()

  Prints the data of the shapes.

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 (xml_node<> *node, std::string name)

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

  Gets the floating point attributes of a node.
- mColor parseColor (xml_node<> *node, std::string color)

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

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

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

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

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

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

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

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

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

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

  Parses the path element.
- Text * parseText (xml_node<> *node, const mColor &fill_color, const mColor &stroke_color, float stroke_width)

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

  Parses the group of elements.

Private Attributes

- SVGElement * root

  The root of the SVG file.

Static Private Attributes

- static Parser * instance = nullptr

  The instance of the Parser.

3.6.1 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 34 of file Parser.hpp.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 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 136 of file Parser.cpp.

```
136                                         {
137     root = parseElements(file_name);
138 }
```

3.6.3 Member Function Documentation

3.6.3.1 getAttribute()

```
std::string Parser::getAttribute (
            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 255 of file Parser.cpp.

```
255                                                                 {
256     if (name == "text") return removeExtraSpaces(node->value());
257     std::string result;
258     if (node->first_attribute(name.c_str()) == NULL) {
259         if (name == "fill")
260             result = "black";
261         else if (name == "stroke" || name == "transform" || name == "rotate" ||
262                  name == "font-style")
263             result = "none";
264         else if (name == "text-anchor")
265             result = "start";
266         else if (name == "fill-rule")
267             result = "nonzero";
268     } else {
269         result = node->first_attribute(name.c_str())->value();
270     }
271     return result;
272 }
```

3.6.3.2 getFloatAttribute()

```
float Parser::getFloatAttribute (
            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 274 of file Parser.cpp.

```
274                                                                 {
275     float result;
276     if (node->first_attribute(name.c_str()) == NULL) {
277         if (name == "stroke-width" || name == "stroke-opacity" ||
278             name == "fill-opacity" || name == "opacity")
279             result = 1;
280         else
281             result = 0;
282     } else {
283         result = std::stof(node->first_attribute(name.c_str())->value());
284     }
285     return result;
286 }
```

3.6.3.3 getTransformOrder()

```
std::vector< std::string > Parser::getTransformOrder (
            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 373 of file Parser.cpp.

```
373                                                                 {
374     std::string transform_tag = getAttribute(node, "transform");
375     std::vector< std::string > order;
376     std::stringstream ss(transform_tag);
377     std::string type;
378     while (ss » type) {
379         if (type.find("translate") != std::string::npos ||
380             type.find("scale") != std::string::npos ||
381             type.find("rotate") != std::string::npos) {
382             while (type.find(")") == std::string::npos) {
383                 std::string temp;
384                 ss » temp;
385                 type += " " + temp;
386             }
387             std::string temp = type.substr(0, type.find("(") + 1);
388             temp.erase(std::remove(temp.begin(), temp.end(), ' '), temp.end());
389             type.erase(0, type.find("(") + 1);
390             type = temp + type;
391             order.push_back(type);
392         }
393     }
394     return order;
395 }
```

3.6.3.4 parseCircle()

```
Circle * Parser::parseCircle (
            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 446 of file Parser.cpp.

```
447                                                                 {
448     float cx = getFloatAttribute(node, "cx");
449     float cy = getFloatAttribute(node, "cy");
450     float radius = getFloatAttribute(node, "r");
```

```
451     Circle *shape = new Circle(radius, Vector2Df(cx, cy), fill_color,
452                                stroke_color, stroke_width);
453     return shape;
454 }
```

3.6.3.5 parseColor()

```
mColor Parser::parseColor (
            xml_node<> * node,
            std::string color )  [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.

Returns

The color attributes of the node.

Definition at line 288 of file Parser.cpp.

```
288                                                         {
289     std::string color = getAttribute(node, name);
290     color.erase(std::remove(color.begin(), color.end(), ' '), color.end());
291     for (auto &c : color) c = tolower(c);
292     if (color == "none")
293         return mColor::Transparent;
294     else {
295         mColor result;
296         if (color.find("#") != std::string::npos) {
297             result = getHexColor(color);
298         } else if (color.find("rgb") != std::string::npos) {
299             result = getRgbColor(color);
300         } else {
301             auto color_code = color_map.find(color);
302             if (color_code == color_map.end()) {
303                 std::cout << "Color " << color << " not found" << std::endl;
304                 exit(-1);
305             }
306             result = color_code->second;
307         }
308 
309         result.a = result.a * getFloatAttribute(node, name + "-opacity") *
310                    getFloatAttribute(node, "opacity");
311         return result;
312     }
313 }
```

3.6.3.6 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 152 of file Parser.cpp.

```
152                                                 {
153     xml_document<> doc;
154     std::ifstream file(file_name);
155     std::vector< char > buffer((std::istreambuf_iterator< char >(file)),
156                                std::istreambuf_iterator< char >());
157     buffer.push_back('\0');
158     doc.parse< 0 >(&buffer[0]);
159 
160     xml_node<> *svg = doc.first_node();
161     xml_node<> *node = svg->first_node();
162     xml_node<> *prev = NULL;
163 
164     SVGElement *root = new Group();
165     SVGElement *current = root;
166 
167     while (node) {
168         if (std::string(node->name()) == "g") {
169             Group *group = dynamic_cast< Group * >(current);
170             for (auto group_attribute : group->getAttributes()) {
171                 bool found = false;
172                 for (auto attribute = node->first_attribute(); attribute;
173                      attribute = attribute->next_attribute()) {
174                     if (std::string(attribute->name()) ==
175                         group_attribute.first) {
176                         if (group_attribute.first == "opacity") {
177                             std::string opacity = std::to_string(
178                                 std::stof(attribute->value()) *
179                                 std::stof(group_attribute.second));
180                             char *value = doc.allocate_string(opacity.c_str());
181                             attribute->value(value);
182                         }
183                         found = true;
184                         break;
185                     }
186                 }
187                 if (!found && group_attribute.first != "transform") {
188                     char *name =
189                         doc.allocate_string(group_attribute.first.c_str());
190                     char *value =
191                         doc.allocate_string(group_attribute.second.c_str());
192                     xml_attribute<> *new_attribute =
193                         doc.allocate_attribute(name, value);
194                     node->append_attribute(new_attribute);
195                 }
196             }
197             Group *new_group = new Group(xmlToString(node->first_attribute()));
198             new_group->setTransforms(getTransformOrder(node));
199             current->addElement(new_group);
200             current = new_group;
201             prev = node;
202             node = node->first_node();
203         } else {
204             Group *group = dynamic_cast< Group * >(current);
205             for (auto group_attribute : group->getAttributes()) {
206                 bool found = false;
207                 for (auto attribute = node->first_attribute(); attribute;
208                      attribute = attribute->next_attribute()) {
209                     if (std::string(attribute->name()) ==
210                         group_attribute.first) {
211                         if (group_attribute.first == "opacity") {
212                             std::string opacity = std::to_string(
213                                 std::stof(attribute->value()) *
214                                 std::stof(group_attribute.second));
215                             char *value = doc.allocate_string(opacity.c_str());
216                             attribute->value(value);
217                         }
218                         found = true;
219                         break;
220                     }
221                 }
222                 if (!found && group_attribute.first != "transform") {
```

```
223                     char *name =
224                         doc.allocate_string(group_attribute.first.c_str());
225                     char *value =
226                         doc.allocate_string(group_attribute.second.c_str());
227                     xml_attribute<> *new_attribute =
228                         doc.allocate_attribute(name, value);
229                     node->append_attribute(new_attribute);
230                 }
231             }
232             SVGElement *shape = parseShape(node);
233             if (shape != NULL) current->addElement(shape);
234             prev = node;
235             node = node->next_sibling();
236         }
237         if (node == NULL && current != root) {
238             while (prev->parent()->next_sibling() == NULL) {
239                 current = current->getParent();
240                 prev = prev->parent();
241                 if (prev == svg) {
242                     break;
243                 }
244             }
245             if (prev == svg) {
246                 break;
247             }
248             current = current->getParent();
249             node = prev->parent()->next_sibling();
250         }
251     }
252     return root;
253 }
```

3.6.3.7 parseEllipse()

```
Ell * Parser::parseEllipse (
            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 456 of file Parser.cpp.

```
457                                                                         {
458     float radius_x = getFloatAttribute(node, "rx");
459     float radius_y = getFloatAttribute(node, "ry");
460     float cx = getFloatAttribute(node, "cx");
461     float cy = getFloatAttribute(node, "cy");
462     Ell *shape = new Ell(Vector2Df(radius_x, radius_y), Vector2Df(cx, cy),
463                          fill_color, stroke_color, stroke_width);
464     return shape;
465 }
```

3.6.3.8 parseLine()

```
Line * Parser::parseLine (
            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 424 of file Parser.cpp.

```
425                                         {
426     Line *shape = new Line(
427         Vector2Df(getFloatAttribute(node, "x1"), getFloatAttribute(node, "y1")),
428         Vector2Df(getFloatAttribute(node, "x2"), getFloatAttribute(node, "y2")),
429         stroke_color, stroke_width);
430     return shape;
431 }
```

3.6.3.9 parsePath()

```
Path * Parser::parsePath (
            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 523 of file Parser.cpp.

```
524                                                                 {
525     Path *shape = new Path(fill_color, stroke_color, stroke_width);
```

```
526     std::vector< PathPoint > points = parsePathPoints(node);
527     for (auto point : points) {
528         shape->addPoint(point);
529     }
530     std::string fill_rule = getAttribute(node, "fill-rule");
531     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
532                     fill_rule.end());
533     shape->setFillRule(fill_rule);
534     return shape;
535 }
```

3.6.3.10 parsePathPoints()

```
std::vector< PathPoint > Parser::parsePathPoints (
            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 331 of file Parser.cpp.

```
331                                                                 {
332     std::vector< PathPoint > points;
333     std::string path_string = getAttribute(node, "d");
334 
335     formatSvgPathString(path_string);
336     std::stringstream ss(path_string);
337     std::string element;
338     PathPoint pPoint{{0, 0}, 'M'};
339 
340     while (ss » element) {
341         if (std::isalpha(element[0])) {
342             pPoint.TC = element[0];
343             if (tolower(pPoint.TC) == 'm' || tolower(pPoint.TC) == 'l' ||
344                 tolower(pPoint.TC) == 'c')
345                 ss » pPoint.Point.x » pPoint.Point.y;
346             else if (tolower(pPoint.TC) == 'h') {
347                 ss » pPoint.Point.x;
348                 pPoint.Point.y = 0;
349             } else if (tolower(pPoint.TC) == 'v') {
350                 ss » pPoint.Point.y;
351                 pPoint.Point.x = 0;
352             }
353         } else {
354             if (tolower(pPoint.TC) == 'm' || tolower(pPoint.TC) == 'l' ||
355                 tolower(pPoint.TC) == 'c') {
356                 if (tolower(pPoint.TC) == 'm') pPoint.TC = 'L';
357                 pPoint.Point.x = std::stof(element);
358                 ss » pPoint.Point.y;
359             } else if (tolower(pPoint.TC) == 'h') {
360                 pPoint.Point.x = std::stof(element);
361                 pPoint.Point.y = 0;
362             } else if (tolower(pPoint.TC) == 'v') {
363                 pPoint.Point.y = std::stof(element);
364                 pPoint.Point.x = 0;
365             }
366         }
367         points.push_back(pPoint);
368     }
369 
370     return points;
371 }
```

3.6.3.11 parsePoints()

```
std::vector< Vector2Df > Parser::parsePoints (
            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 315 of file Parser.cpp.

```
315                                                                 {
316     std::vector< Vector2Df > points;
317     std::string points_string = getAttribute(node, "points");
318
319     std::stringstream ss(points_string);
320     float x, y;
321
322     while (ss » x) {
323         if (ss.peek() == ',') ss.ignore();
324         ss » y;
325         points.push_back(Vector2Df(x, y));
326     }
327
328     return points;
329 }
```

3.6.3.12 parsePolygon()

```
Plygon * Parser::parsePolygon (
            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 467 of file Parser.cpp.

```
468                                                                                 {
469     Plygon *shape = new Plygon(fill_color, stroke_color, stroke_width);
470     std::vector< Vector2Df > points = parsePoints(node);
471     for (auto point : points) {
472         shape->addPoint(point);
473     }
474     std::string fill_rule = getAttribute(node, "fill-rule");
475     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
476                     fill_rule.end());
477     shape->setFillRule(fill_rule);
478     return shape;
479 }
```

3.6.3.13 parsePolyline()

```
Plyline * Parser::parsePolyline (
            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 481 of file Parser.cpp.

```
482                                                                                 {
483     Plyline *shape = new Plyline(fill_color, stroke_color, stroke_width);
484     std::vector< Vector2Df > points = parsePoints(node);
485     for (auto point : points) {
486         shape->addPoint(point);
487     }
488     std::string fill_rule = getAttribute(node, "fill-rule");
489     fill_rule.erase(std::remove(fill_rule.begin(), fill_rule.end(), ' '),
490                     fill_rule.end());
491     shape->setFillRule(fill_rule);
492     return shape;
493 }
```

3.6.3.14 parseRect()

```
Rect * Parser::parseRect (
            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 433 of file Parser.cpp.

```
434                                                                                 {
435     float x = getFloatAttribute(node, "x");
436     float y = getFloatAttribute(node, "y");
437     float rx = getFloatAttribute(node, "rx");
438     float ry = getFloatAttribute(node, "ry");
439     Rect *shape =
440         new Rect(getFloatAttribute(node, "width"),
441                  getFloatAttribute(node, "height"), Vector2Df(x, y),
442                  Vector2Df(rx, ry), fill_color, stroke_color, stroke_width);
443     return shape;
444 }
```

3.6.3.15 parseShape()

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

Parses the group of elements.

Parameters

node	The node to be parsed.

Returns

The group of elements

Definition at line 397 of file Parser.cpp.

```
397                                                 {
398     SVGElement *shape = NULL;
399     std::string type = node->name();
400     mColor stroke_color = parseColor(node, "stroke");
401     mColor fill_color = parseColor(node, "fill");
402     float stroke_width = getFloatAttribute(node, "stroke-width");
403     if (type == "line") {
404         shape = parseLine(node, stroke_color, stroke_width);
405     } else if (type == "rect") {
406         shape = parseRect(node, fill_color, stroke_color, stroke_width);
407     } else if (type == "circle") {
408         shape = parseCircle(node, fill_color, stroke_color, stroke_width);
409     } else if (type == "ellipse") {
410         shape = parseEllipse(node, fill_color, stroke_color, stroke_width);
411     } else if (type == "polygon") {
412         shape = parsePolygon(node, fill_color, stroke_color, stroke_width);
413     } else if (type == "polyline") {
414         shape = parsePolyline(node, fill_color, stroke_color, stroke_width);
415     } else if (type == "path") {
416         shape = parsePath(node, fill_color, stroke_color, stroke_width);
```

```
417     } else if (type == "text") {
418         return parseText(node, fill_color, stroke_color, stroke_width);
419     }
420     if (shape != NULL) shape->setTransforms(getTransformOrder(node));
421     return shape;
422 }
```

3.6.3.16 parseText()

```
Text * Parser::parseText (
            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 495 of file Parser.cpp.

```
496                                                                         {
497     float x = getFloatAttribute(node, "x");
498     float y = getFloatAttribute(node, "y");
499     float font_size = getFloatAttribute(node, "font-size");
500     std::string text = getAttribute(node, "text");
501
502     Text *shape = new Text(Vector2Df(x - 7, y - font_size + 5), text, font_size,
503                            fill_color, stroke_color, stroke_width);
504
505     std::string anchor = getAttribute(node, "text-anchor");
506     anchor.erase(std::remove(anchor.begin(), anchor.end(), ' '), anchor.end());
507     shape->setAnchor(anchor);
508
509     std::string style = getAttribute(node, "font-style");
510     style.erase(std::remove(style.begin(), style.end(), ' '), style.end());
511     shape->setFontStyle(style);
512
513     float dx = getFloatAttribute(node, "dx");
514     float dy = getFloatAttribute(node, "dy");
515     std::string transform =
516         "translate(" + std::to_string(dx) + " " + std::to_string(dy) + ")";
517     std::vector< std::string > transform_order = getTransformOrder(node);
518     transform_order.push_back(transform);
519     shape->setTransforms(transform_order);
520     return shape;
521 }
```

3.6.3.17 printShapesData()

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

Prints the data of the shapes.

Note

This function is used for debugging.

Definition at line 539 of file Parser.cpp.

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

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

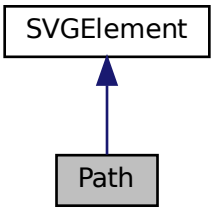
- src/Parser.hpp
- src/Parser.cpp

3.7 Path Class Reference

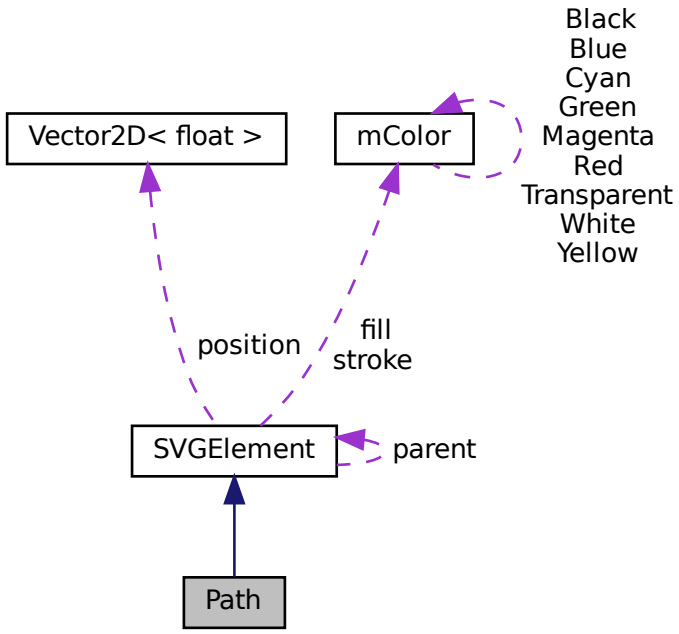
Represents a path element in 2D space.

`#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

3.7.1 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 24 of file Path.hpp.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 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.

```
3                                                                    {
4     setFillColor(fill);
5     setOutlineColor(stroke);
6     setOutlineThickness(stroke_width);
7 }
```

3.7.3 Member Function Documentation

3.7.3.1 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 11 of file Path.cpp.

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

3.7.3.2 getClass()

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

Gets the type of the shape.

Returns

The string "Path".

Implements SVGElement.

Definition at line 9 of file Path.cpp.

```
9 { return "Path"; }
```

3.7.3.3 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 17 of file Path.cpp.

```
17 { return fill_rule; }
```

3.7.3.4 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 13 of file Path.cpp.

```
13 { return points; }
```

3.7.3.5 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 19 of file Path.cpp.

```
19                              {
20     SVGElement::printData();
21     std::cout « "Points: ";
22     for (auto point : points) {
23         std::cout « point.TC « " " « point.Point.x « " " « point.Point.y
24                   « " ";
25     }
26 }
```

3.7.3.6 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 15 of file Path.cpp.

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

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

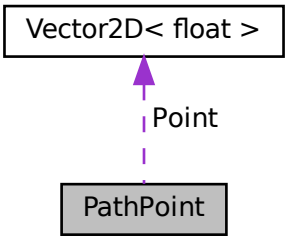
- src/graphics/Path.hpp
- src/graphics/Path.cpp

3.8 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**

3.8.1 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:

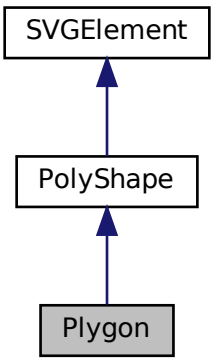
- src/graphics/Path.hpp

3.9 Plygon Class Reference

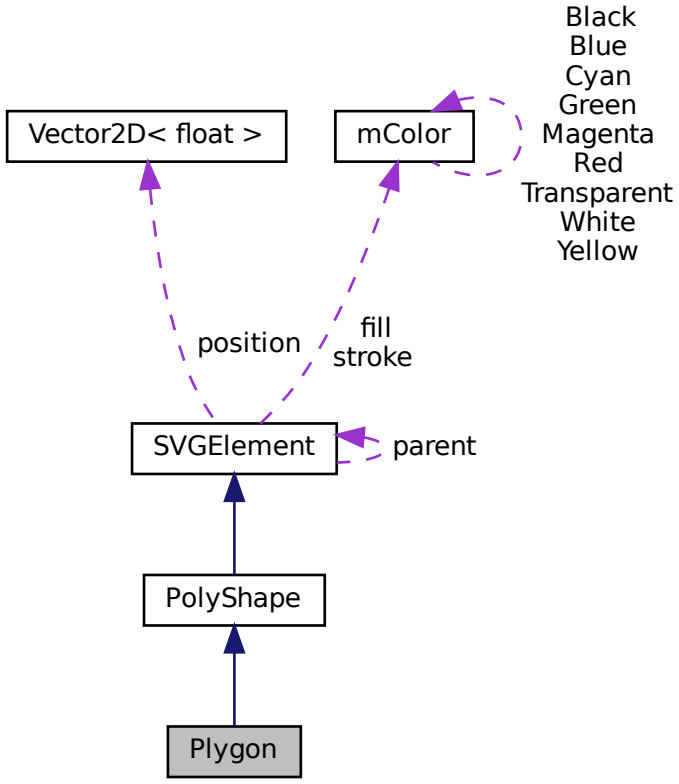
Represents a polygon in 2D space.

`#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.
- void setFillRule (std::string fill_rule)

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

  Gets the fill rule of the polygon.

Private Attributes

- std::string fill_rule

  Fill rule of the polygon.

Additional Inherited Members

3.9.1 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.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 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) {}
```

3.9.3 Member Function Documentation

3.9.3.1 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"; }
```

3.9.3.2 getFillRule()

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

Gets the fill rule of the polygon.

Returns

The fill rule of the polygon.

Definition at line 10 of file Polygon.cpp.

```
10 { return fill_rule; }
```

3.9.3.3 setFillRule()

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

Sets the fill rule of the polygon.

Parameters

fill_rule	The new fill rule of the polygon.

Definition at line 8 of file Polygon.cpp.

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

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

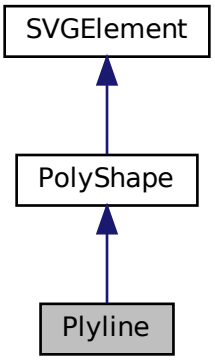
- src/graphics/Polygon.hpp
- src/graphics/Polygon.cpp

3.10 Plyline Class Reference

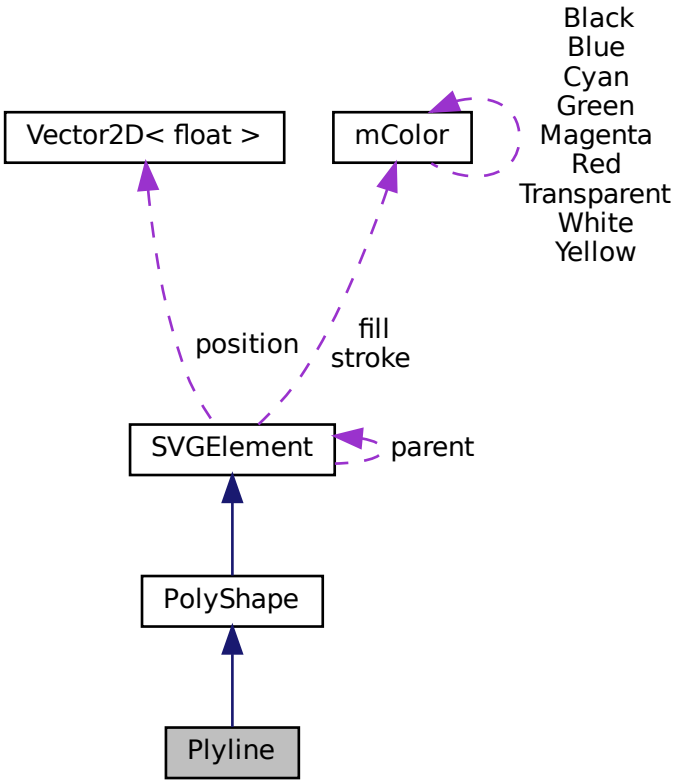
Represents a polyline in 2D space.

```
#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.
- void setFillRule (std::string fill_rule)

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

  Gets the fill rule of the polyline.

Private Attributes

- std::string fill_rule

  Fill rule of the polyline.

Additional Inherited Members

3.10.1 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.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 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) {}
```

3.10.3 Member Function Documentation

3.10.3.1 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"; }
```

3.10.3.2 getFillRule()

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

Gets the fill rule of the polyline.

Returns

The fill rule of the polyline.

Definition at line 12 of file Polyline.cpp.

```
12 { return fill_rule; }
```

3.10.3.3 setFillRule()

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

Sets the fill rule of the polyline.

Parameters

fill_rule	The new fill rule of the polyline.

Definition at line 8 of file Polyline.cpp.

```
8                                          {
```

```
9     this->fill_rule = fill_rule;
10 }
```

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

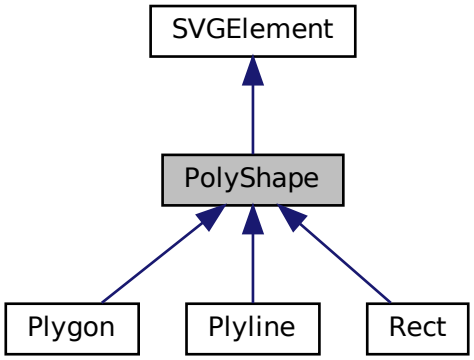
- src/graphics/Polyline.hpp
- src/graphics/Polyline.cpp

3.11 PolyShape Class Reference

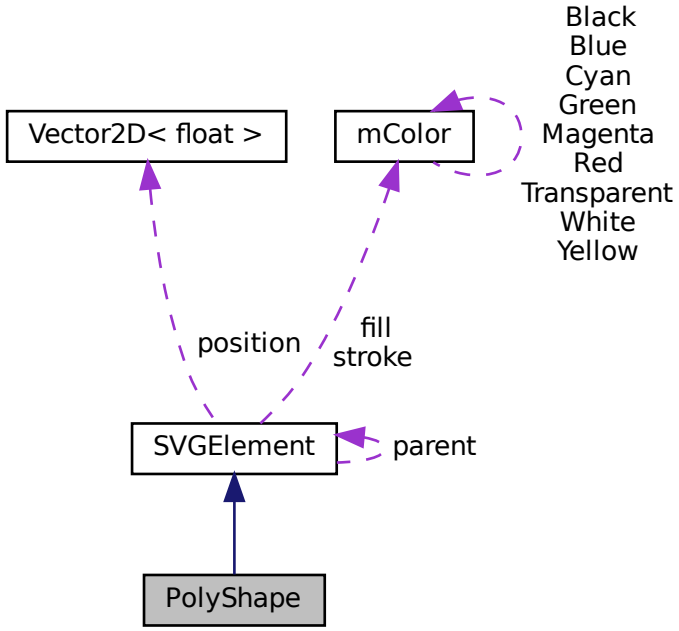
Abstract base class for polygon and polyline shapes in 2D space.

`#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 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.

3.11.1 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.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 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.

```
4                                                   {
5     setFillColor(fill);
6     setOutlineColor(stroke);
7     setOutlineThickness(stroke_width);
8 }
```

3.11.3 Member Function Documentation

3.11.3.1 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 10 of file PolyShape.cpp.

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

3.11.3.2 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 Rect, Plyline, and Plygon.

3.11.3.3 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 12 of file PolyShape.cpp.

```
12 { return points; }
```

3.11.3.4 printData()

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

Prints the data of the shape.

Note

This function is used for debugging purposes.

Reimplemented from SVGElement.

Reimplemented in Rect.

Definition at line 14 of file PolyShape.cpp.

```
14                                  {
15     SVGElement::printData();
16     std::cout « "Points: ";
17     for (auto& point : getPoints()) {
18         std::cout « point.x « "," « point.y « " ";
19     }
20     std::cout « std::endl;
21 }
```

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

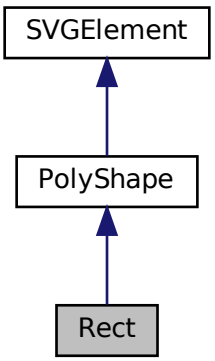
- src/graphics/PolyShape.hpp
- src/graphics/PolyShape.cpp

3.12 Rect Class Reference

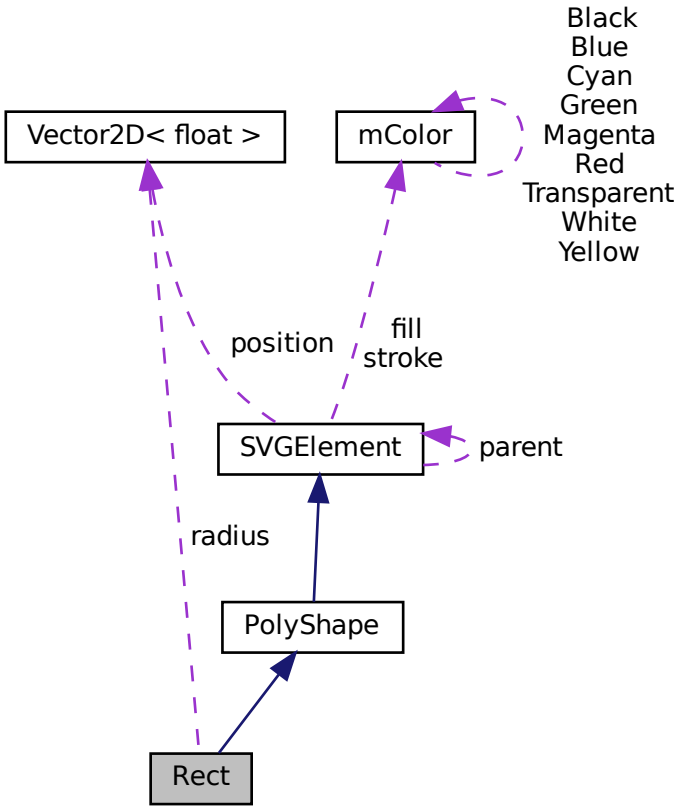
Represents a rectangle in 2D space.

`#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

3.12.1 Detailed Description

Represents a rectangle in 2D space.

The Rect class is derived from the PolyShape 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.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 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     : PolyShape(fill, stroke, stroke_width), width(width), height(height),
6       radius(radius) {
7     addPoint(Vector2Df(0, 0));
8     addPoint(Vector2Df(width, 0));
9     addPoint(Vector2Df(width, height));
10     addPoint(Vector2Df(0, height));
11     setPosition(position);
12 }
```

3.12.3 Member Function Documentation

3.12.3.1 getClass()

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

Gets the type of the shape.

Returns

The string "Rect".

Implements PolyShape.

Definition at line 14 of file Rect.cpp.

```
14 { return "Rect"; }
```

3.12.3.2 getHeight()

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

Gets the height of the rectangle.

Returns

The height of the rectangle.

Definition at line 30 of file Rect.cpp.

```
30 { return height; }
```

3.12.3.3 getRadius()

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

Gets the radii of the rectangle.

Returns

The radii of the rectangle.

Definition at line 34 of file Rect.cpp.

```
34 { return radius; }
```

3.12.3.4 getWidth()

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

Gets the width of the rectangle.

Returns

The width of the rectangle.

Definition at line 22 of file Rect.cpp.

```
22 { return width; }
```

3.12.3.5 printData()

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

Prints the data of the rectangle.

Note

This function is used for debugging purposes.

Reimplemented from PolyShape.

Definition at line 36 of file Rect.cpp.

```
36                              {
37     SVGElement::printData();
38     std::cout « "Width: " « getWidth() « std::endl;
39     std::cout « "Height: " « getHeight() « std::endl;
40     std::cout « "Radius: " « getRadius().x « " " « getRadius().y
41               « std::endl;
42 }
```

3.12.3.6 setHeight()

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

Sets the height of the rectangle.

Parameters

height	The new height of the rectangle.

Definition at line 24 of file Rect.cpp.

```
24                                  {
25     this->height = height;
26     points[2].y = height;
27     points[3].y = height;
```

```
28 }
```

3.12.3.7 setRadius()

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

Sets the radii of the rectangle.

Parameters

radius	The new radii of the rectangle.

Definition at line 32 of file Rect.cpp.

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

3.12.3.8 setWidth()

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

Sets the width of the rectangle.

Parameters

width	The new width of the rectangle.

Definition at line 16 of file Rect.cpp.

```
16                                {
17     this->width = width;
18     points[1].x = width;
19     points[2].x = width;
20 }
```

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

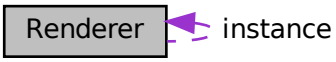
- src/graphics/Rect.hpp
- src/graphics/Rect.cpp

3.13 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.
- Renderer ()

  Private constructor for the Renderer class.

Static Private Attributes

- static Renderer * instance = nullptr

  Singleton instance of the Renderer class.

3.13.1 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.

3.13.2 Member Function Documentation

3.13.2.1 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 46 of file Renderer.cpp.

```
47                                                                  {
48     for (auto type : transform_order) {
49         if (type.find("translate") != std::string::npos) {
50             float trans_x = getTranslate(type).first,
51                   trans_y = getTranslate(type).second;
52             graphics.TranslateTransform(trans_x, trans_y);
53         } else if (type.find("rotate") != std::string::npos) {
54             float degree = getRotate(type);
55             graphics.RotateTransform(degree);
56         } else if (type.find("scale") != std::string::npos) {
57             if (type.find(",") != std::string::npos) {
58                 float scale_x = getScaleXY(type).first,
59                       scale_y = getScaleXY(type).second;
60                 graphics.ScaleTransform(scale_x, scale_y);
61             } else {
62                 float scale = getScale(type);
63                 graphics.ScaleTransform(scale, scale);
64             }
65         }
66     }
67 }
```

3.13.2.2 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 69 of file Renderer.cpp.

```
69                                                                          {
70     for (auto shape : group->getElements()) {
71         Gdiplus::Matrix original;
72         graphics.GetTransform(&original);
73         applyTransform(shape->getTransforms(), graphics);
74         if (shape->getClass() == "Group") {
75             Group* group = dynamic_cast< Group* >(shape);
76             draw(graphics, group);
77         } else if (shape->getClass() == "Polyline") {
78             Plyline* polyline = dynamic_cast< Plyline* >(shape);
79             drawPolyline(graphics, polyline);
80         } else if (shape->getClass() == "Text") {
81             Text* text = dynamic_cast< Text* >(shape);
82             drawText(graphics, text);
83         } else if (shape->getClass() == "Rect") {
84             Rect* rectangle = dynamic_cast< Rect* >(shape);
85             drawRectangle(graphics, rectangle);
86         } else if (shape->getClass() == "Circle") {
87             Circle* circle = dynamic_cast< Circle* >(shape);
88             drawCircle(graphics, circle);
89         } else if (shape->getClass() == "Ellipse") {
90             Ell* ellipse = dynamic_cast< Ell* >(shape);
91             drawEllipse(graphics, ellipse);
92         } else if (shape->getClass() == "Line") {
93             Line* line = dynamic_cast< Line* >(shape);
94             drawLine(graphics, line);
95         } else if (shape->getClass() == "Polygon") {
96             Plygon* polygon = dynamic_cast< Plygon* >(shape);
97             drawPolygon(graphics, polygon);
98         } else if (shape->getClass() == "Path") {
99             Path* path = dynamic_cast< Path* >(shape);
100             drawPath(graphics, path);
101         }
102         graphics.SetTransform(&original);
103     }
104 }
```

3.13.2.3 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 145 of file Renderer.cpp.

```
145                                                                         {
146     mColor fill_color = circle->getFillColor();
147     mColor outline_color = circle->getOutlineColor();
148     Gdiplus::Pen circleOutline(Gdiplus::Color(outline_color.a, outline_color.r,
149                                               outline_color.g, outline_color.b),
150                                circle->getOutlineThickness());
151     Gdiplus::SolidBrush circleFill(
152         Gdiplus::Color(fill_color.a, fill_color.r, fill_color.g, fill_color.b));
153     graphics.FillEllipse(&circleFill,
154                          circle->getPosition().x - circle->getRadius().x,
155                          circle->getPosition().y - circle->getRadius().y,
156                          circle->getRadius().x * 2, circle->getRadius().y * 2);
157     graphics.DrawEllipse(&circleOutline,
158                          circle->getPosition().x - circle->getRadius().x,
159                          circle->getPosition().y - circle->getRadius().y,
160                          circle->getRadius().x * 2, circle->getRadius().x * 2);
161 }
```

3.13.2.4 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 163 of file Renderer.cpp.

```
163                                                                         {
164     mColor fill_color = ellipse->getFillColor();
165     mColor outline_color = ellipse->getOutlineColor();
166     Gdiplus::Pen ellipseOutline(
167         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
168                        outline_color.b),
169         ellipse->getOutlineThickness());
170     Gdiplus::SolidBrush ellipseFill(
171         Gdiplus::Color(fill_color.a, fill_color.r, fill_color.g, fill_color.b));
172     graphics.FillEllipse(
173         &ellipseFill, ellipse->getPosition().x - ellipse->getRadius().x,
174         ellipse->getPosition().y - ellipse->getRadius().y,
175         ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
176     graphics.DrawEllipse(
177         &ellipseOutline, ellipse->getPosition().x - ellipse->getRadius().x,
178         ellipse->getPosition().y - ellipse->getRadius().y,
179         ellipse->getRadius().x * 2, ellipse->getRadius().y * 2);
180 }
```

3.13.2.5 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 106 of file Renderer.cpp.

```
106                                                                     {
107     mColor color = line->getOutlineColor();
108     Gdiplus::Pen linePen(Gdiplus::Color(color.a, color.r, color.g, color.b),
109                          line->getOutlineThickness());
110     Gdiplus::PointF startPoint(line->getPosition().x, line->getPosition().y);
111     Gdiplus::PointF endPoint(line->getDirection().x, line->getDirection().y);
112     graphics.DrawLine(&linePen, startPoint, endPoint);
113 }
```

3.13.2.6 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 286 of file Renderer.cpp.

```
286                                                                     {
287     mColor outline_color = path->getOutlineColor();
288     mColor fill_color = path->getFillColor();
289     Gdiplus::Pen pathPen(Gdiplus::Color(outline_color.a, outline_color.r,
290                                         outline_color.g, outline_color.b),
291                          path->getOutlineThickness());
292     Gdiplus::SolidBrush pathFill(
293         Gdiplus::Color(fill_color.a, fill_color.r, fill_color.g, fill_color.b));
294
295     Gdiplus::FillMode fillMode;
296     if (path->getFillRule() == "evenodd") {
297         fillMode = Gdiplus::FillModeAlternate;
298     } else if (path->getFillRule() == "nonzero") {
299         fillMode = Gdiplus::FillModeWinding;
300     }
301     Gdiplus::GraphicsPath gdiPath(fillMode);
302
303     const std::vector< PathPoint >& points = path->getPoints();
304     int n = points.size();
305     Vector2Df firstPoint{0, 0}, curPoint{0, 0};
306
307     for (int i = 0; i < n; ++i) {
308         if (points[i].TC == 'M') {
309             firstPoint = points[i].Point;
310             gdiPath.StartFigure();
311             curPoint = firstPoint;
312         } else if (points[i].TC == 'm') {
313             firstPoint.x = curPoint.x + points[i].Point.x;
314             firstPoint.y = curPoint.y + points[i].Point.y;
315             gdiPath.StartFigure();
316             curPoint = firstPoint;
317         } else if (points[i].TC == 'L') {
318             gdiPath.AddLine(curPoint.x, curPoint.y, points[i].Point.x,
319                             points[i].Point.y);
```

```
320             curPoint = points[i].Point;
321         } else if (points[i].TC == 'l') {
322             Vector2Df endPoint{curPoint.x + points[i].Point.x,
323                                curPoint.y + points[i].Point.y};
324             gdiPath.AddLine(curPoint.x, curPoint.y, endPoint.x, endPoint.y);
325             curPoint = endPoint;
326         } else if (points[i].TC == 'H') {
327             Vector2Df endPoint{points[i].Point.x, curPoint.y};
328             gdiPath.AddLine(curPoint.x, curPoint.y, endPoint.x, endPoint.y);
329             curPoint = endPoint;
330         } else if (points[i].TC == 'h') {
331             Vector2Df endPoint{curPoint.x + points[i].Point.x, curPoint.y};
332             gdiPath.AddLine(curPoint.x, curPoint.y, endPoint.x, endPoint.y);
333             curPoint = endPoint;
334         } else if (points[i].TC == 'V') {
335             Vector2Df endPoint{curPoint.x, points[i].Point.y};
336             gdiPath.AddLine(curPoint.x, curPoint.y, endPoint.x, endPoint.y);
337             curPoint = endPoint;
338         } else if (points[i].TC == 'v') {
339             Vector2Df endPoint{curPoint.x, curPoint.y + points[i].Point.y};
340             gdiPath.AddLine(curPoint.x, curPoint.y, endPoint.x, endPoint.y);
341             curPoint = endPoint;
342         } else if (points[i].TC == 'C') {
343             if (i + 2 < n) {
344                 Vector2Df controlPoint1 = points[i].Point;
345                 Vector2Df controlPoint2 = points[i + 1].Point;
346                 Vector2Df controlPoint3 = points[i + 2].Point;
347                 gdiPath.AddBezier(curPoint.x, curPoint.y, controlPoint1.x,
348                                   controlPoint1.y, controlPoint2.x,
349                                   controlPoint2.y, controlPoint3.x,
350                                   controlPoint3.y);
351                 i += 2;
352                 curPoint = controlPoint3;
353             }
354         } else if (points[i].TC == 'c') {
355             if (i + 2 < n) {
356                 Vector2Df controlPoint1 =
357                     Vector2Df{curPoint.x + points[i].Point.x,
358                               curPoint.y + points[i].Point.y};
359                 Vector2Df controlPoint2 =
360                     Vector2Df{curPoint.x + points[i + 1].Point.x,
361                               curPoint.y + points[i + 1].Point.y};
362                 Vector2Df controlPoint3 =
363                     Vector2Df{curPoint.x + points[i + 2].Point.x,
364                               curPoint.y + points[i + 2].Point.y};
365                 gdiPath.AddBezier(curPoint.x, curPoint.y, controlPoint1.x,
366                                   controlPoint1.y, controlPoint2.x,
367                                   controlPoint2.y, controlPoint3.x,
368                                   controlPoint3.y);
369                 i += 2;
370                 curPoint = controlPoint3;
371             }
372         } else if (points[i].TC == 'Z' || points[i].TC == 'z') {
373             gdiPath.CloseFigure();
374             curPoint = firstPoint;
375         }
376     }
377     graphics.FillPath(&pathFill, &gdiPath);
378     graphics.DrawPath(&pathPen, &gdiPath);
379 }
```

3.13.2.7 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 182 of file Renderer.cpp.

```
182                                                                                 {
183     mColor fill_color = polygon->getFillColor();
184     mColor outline_color = polygon->getOutlineColor();
185     Gdiplus::Pen polygonOutline(
186         Gdiplus::Color(outline_color.a, outline_color.r, outline_color.g,
187                        outline_color.b),
188         polygon->getOutlineThickness());
189     Gdiplus::SolidBrush polygonFill(
190         Gdiplus::Color(fill_color.a, fill_color.r, fill_color.g, fill_color.b));
191
192     Gdiplus::PointF* points = new Gdiplus::PointF[polygon->getPoints().size()];
193     int idx = 0;
194     const std::vector< Vector2Df >& vertices = polygon->getPoints();
195     for (const Vector2Df vertex : vertices) {
196         points[idx++] = Gdiplus::PointF(vertex.x, vertex.y);
197     }
198
199     Gdiplus::FillMode fillMode;
200     if (polygon->getFillRule() == "evenodd") {
201         fillMode = Gdiplus::FillModeAlternate;
202     } else if (polygon->getFillRule() == "nonzero") {
203         fillMode = Gdiplus::FillModeWinding;
204     }
205     graphics.FillPolygon(&polygonFill, points, idx, fillMode);
206     graphics.DrawPolygon(&polygonOutline, points, idx);
207     delete[] points;
208 }
```

3.13.2.8 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 254 of file Renderer.cpp.

```
255                                                 {
256     mColor outline_color = polyline->getOutlineColor();
257     mColor fill_color = polyline->getFillColor();
258     Gdiplus::Pen polylinePen(Gdiplus::Color(outline_color.a, outline_color.r,
259                                             outline_color.g, outline_color.b),
260                              polyline->getOutlineThickness());
261     Gdiplus::SolidBrush polylineFill(
262         Gdiplus::Color(fill_color.a, fill_color.r, fill_color.g, fill_color.b));
263
264     Gdiplus::FillMode fillMode;
265     if (polyline->getFillRule() == "evenodd") {
266         fillMode = Gdiplus::FillModeAlternate;
267     } else if (polyline->getFillRule() == "nonzero") {
268         fillMode = Gdiplus::FillModeWinding;
269     }
270     Gdiplus::GraphicsPath path(fillMode);
271     const std::vector< Vector2Df >& points = polyline->getPoints();
272     if (points.size() < 2) {
273         return;
274     }
275
276     path.StartFigure();
277     path.AddLine(points[0].x, points[0].y, points[1].x, points[1].y);
278     for (size_t i = 2; i < points.size(); ++i) {
279         path.AddLine(points[i - 1].x, points[i - 1].y, points[i].x,
280                      points[i].y);
281     }
282     graphics.FillPath(&polylineFill, &path);
```

```
283     graphics.DrawPath(&polylinePen, &path);
284 }
```

3.13.2.9 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 115 of file Renderer.cpp.

```
116                                                 {
117     float x = rectangle->getPosition().x;
118     float y = rectangle->getPosition().y;
119     float width = rectangle->getWidth();
120     float height = rectangle->getHeight();
121     mColor fill_color = rectangle->getFillColor();
122     mColor outline_color = rectangle->getOutlineColor();
123     Gdiplus::Pen RectOutline(Gdiplus::Color(outline_color.a, outline_color.r,
124                                             outline_color.g, outline_color.b),
125                              rectangle->getOutlineThickness());
126     Gdiplus::SolidBrush RectFill(
127         Gdiplus::Color(fill_color.a, fill_color.r, fill_color.g, fill_color.b));
128     if (rectangle->getRadius().x != 0 || rectangle->getRadius().y != 0) {
129         float dx = rectangle->getRadius().x * 2;
130         float dy = rectangle->getRadius().y * 2;
131         Gdiplus::GraphicsPath path;
132         path.AddArc(x, y, dx, dy, 180, 90);
133         path.AddArc(x + width - dx, y, dx, dy, 270, 90);
134         path.AddArc(x + width - dx, y + height - dy, dx, dy, 0, 90);
135         path.AddArc(x, y + height - dy, dx, dy, 90, 90);
136         path.CloseFigure();
137         graphics.FillPath(&RectFill, &path);
138         graphics.DrawPath(&RectOutline, &path);
139     } else {
140         graphics.FillRectangle(&RectFill, x, y, width, height);
141         graphics.DrawRectangle(&RectOutline, x, y, width, height);
142     }
143 }
```

3.13.2.10 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 212 of file Renderer.cpp.

```
212                                                                         {
213     mColor outline_color = text->getOutlineColor();
214     mColor fill_color = text->getFillColor();
215 
216     graphics.SetTextRenderingHint(Gdiplus::TextRenderingHintAntiAliasGridFit);
217 
218     Gdiplus::SolidBrush textFill(
219         Gdiplus::Color(fill_color.a, fill_color.r, fill_color.g, fill_color.b));
220 
221     Gdiplus::Pen textOutline(Gdiplus::Color(outline_color.a, outline_color.r,
222                                             outline_color.g, outline_color.b),
223                              text->getOutlineThickness());
224 
225     Gdiplus::FontFamily fontFamily(L"Times New Roman");
226 
227     Gdiplus::PointF position(text->getPosition().x, text->getPosition().y);
228     Gdiplus::GraphicsPath path;
229 
230     std::wstring_convert< std::codecvt_utf8_utf16< wchar_t > > converter;
231     std::wstring wideContent = converter.from_bytes(text->getContent());
232     Gdiplus::StringFormat stringFormat;
233     if (text->getAnchor() == "middle") {
234         stringFormat.SetAlignment(Gdiplus::StringAlignmentCenter);
235         position.X += 7;
236     } else if (text->getAnchor() == "end") {
237         stringFormat.SetAlignment(Gdiplus::StringAlignmentFar);
238         position.X += 14;
239     } else {
240         stringFormat.SetAlignment(Gdiplus::StringAlignmentNear);
241     }
242     Gdiplus::FontStyle fontStyle = Gdiplus::FontStyleRegular;
243     if (text->getFontStyle() == "italic" || text->getFontStyle() == "oblique") {
244         fontStyle = Gdiplus::FontStyleItalic;
245         position.Y -= 1;
246     }
247 
248     path.AddString(wideContent.c_str(), wideContent.size(), &fontFamily,
249                    fontStyle, text->getFontSize(), position, &stringFormat);
250     graphics.FillPath(&textFill, &path);
251     graphics.DrawPath(&textOutline, &path);
252 }
```

3.13.2.11 getInstance()

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

Gets the singleton instance of the Renderer class.

Returns

The singleton instance of the Renderer class.

Definition at line 7 of file Renderer.cpp.

```
7                                   {
8     if (instance == nullptr) {
9         instance = new Renderer();
10     }
11     return instance;
12 }
```

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

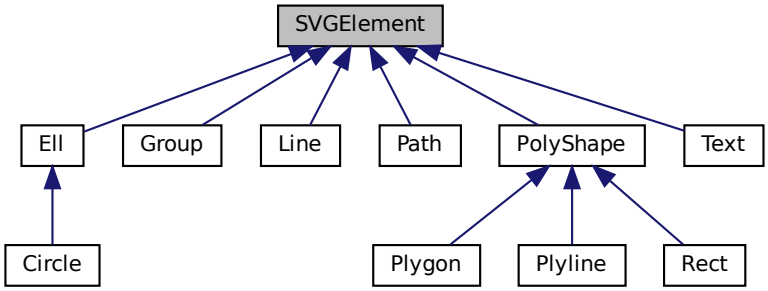
- src/Renderer.hpp
- src/Renderer.cpp

3.14 SVGElement Class Reference

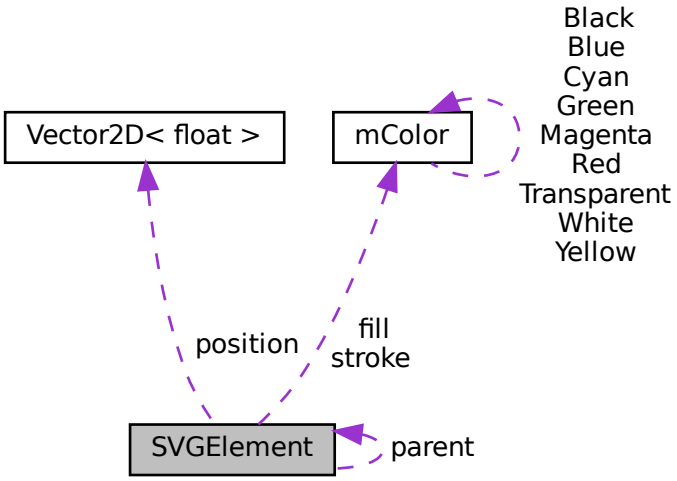
Represents an element in an SVG file.

`#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 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.
- virtual void addElement (SVGElement *element)

  Adds a shape to the composite group.

Protected Member Functions

- SVGElement ()

  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.

3.14.1 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 17 of file SVGElement.hpp.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 SVGElement()

```
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) {}
```

3.14.3 Member Function Documentation

3.14.3.1 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 59 of file SVGElement.cpp.

```
59 {}
```

3.14.3.2 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.

3.14.3.3 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 10 of file SVGElement.cpp.

```
10 { return fill; }
```

3.14.3.4 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 14 of file SVGElement.cpp.

```
14 { return stroke; }
```

3.14.3.5 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 20 of file SVGElement.cpp.

```
20 { return stroke_width; }
```

3.14.3.6 getParent()

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

Parent pointer getter.

Returns

The parent pointer

Note

This function is used for composite design pattern

Definition at line 57 of file SVGElement.cpp.

```
57 { return parent; }
```

3.14.3.7 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 31 of file SVGElement.cpp.

```
31 { return position; }
```

3.14.3.8 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 51 of file SVGElement.cpp.

```
51                                                      {
52     return transforms;
53 }
```

3.14.3.9 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 33 of file SVGElement.cpp.

```
33                                  {
34     std::cout « "Shape: " « getClass() « std::endl;
35     std::cout « "Fill: " « getFillColor() « std::endl;
36     std::cout « "Stroke: " « getOutlineColor() « std::endl;
37     std::cout « "Stroke width: " « getOutlineThickness() « std::endl;
38     std::cout « "Position: " « getPosition().x « " " « getPosition().y
39               « std::endl;
40     std::cout « "Transforms: ";
41     for (auto transform : transforms) {
42         std::cout « transform « " ";
43     }
44     std::cout « std::endl;
45 }
```

3.14.3.10 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 8 of file SVGElement.cpp.

```
8 { fill = color; }
```

3.14.3.11 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 12 of file SVGElement.cpp.

```
12 { stroke = color; }
```

3.14.3.12 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 16 of file SVGElement.cpp.

```
16                                                  {
17     stroke_width = thickness;
18 }
```

3.14.3.13 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 55 of file SVGElement.cpp.

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

3.14.3.14 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 27 of file SVGElement.cpp.

```
27                                                  {
28     setPosition(position.x, position.y);
29 }
```

3.14.3.15 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 22 of file SVGElement.cpp.

```
22                                              {
23     position.x = x;
24     position.y = y;
25 }
```

3.14.3.16 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 47 of file SVGElement.cpp.

```
47                                                                          {
48     this->transforms = transforms;
49 }
```

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

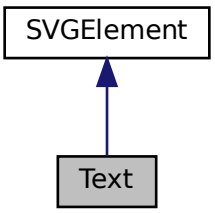
- src/graphics/SVGElement.hpp
- src/graphics/SVGElement.cpp

3.15 Text Class Reference

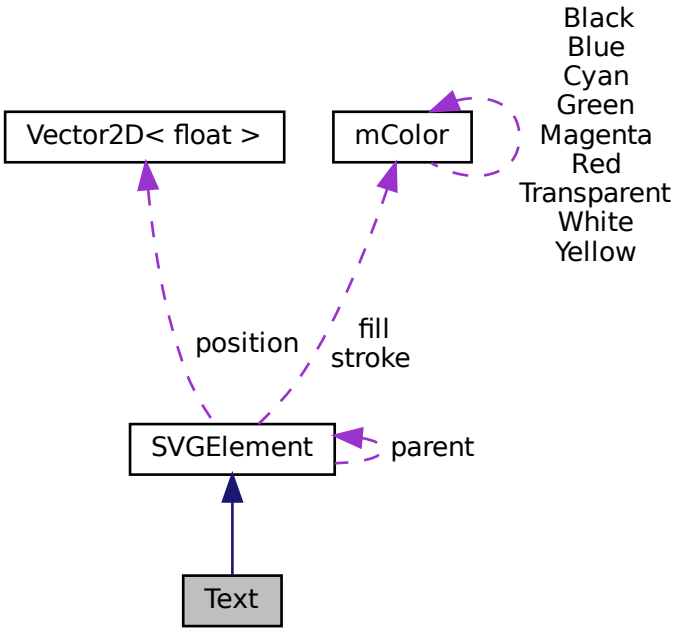
Represents text in 2D space.

`#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

3.15.1 Detailed Description

Represents text in 2D space.

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

Definition at line 12 of file Text.hpp.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 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     : content(text), font_size(font_size) {
6     setOutlineColor(stroke);
7     setOutlineThickness(stroke_width);
8     setFillColor(fill);
9     setPosition(pos);
10 }
```

3.15.3 Member Function Documentation

3.15.3.1 getAnchor()

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

Gets the anchor of the text.

Returns

The anchor of the text.

Definition at line 24 of file Text.cpp.

```
24 { return anchor; }
```

3.15.3.2 getClass()

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

Gets the type of the shape.

Returns

The string "Text".

Implements SVGElement.

Definition at line 12 of file Text.cpp.

```
12 { return "Text"; }
```

3.15.3.3 getContent()

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

Gets the string of the text.

Returns

The string of the text.

Definition at line 20 of file Text.cpp.

```
20 { return content; }
```

3.15.3.4 getFontSize()

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

Gets the font size of the text.

Returns

The font size of the text.

Definition at line 16 of file Text.cpp.

```
16 { return font_size; }
```

3.15.3.5 getFontStyle()

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

Gets the style of the text.

Returns

The style of the text.

Definition at line 28 of file Text.cpp.

```
28 { return style; }
```

3.15.3.6 setAnchor()

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

Sets the anchor of the text.

Parameters

anchor	The new anchor of the text.

Definition at line 22 of file Text.cpp.

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

3.15.3.7 setContent()

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

Sets the string of the text.

Parameters

content	The new string of the text.

Definition at line 18 of file Text.cpp.

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

3.15.3.8 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 14 of file Text.cpp.

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

3.15.3.9 setFontStyle()

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

Sets the style of the text.

Parameters

style	The new style of the text.

Definition at line 26 of file Text.cpp.

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

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

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

3.16 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.

3.16.1 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.

3.16.2 Constructor & Destructor Documentation

3.16.2.1 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) {}
```

3.16.2.2 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) {}
```

3.16.2.3 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:

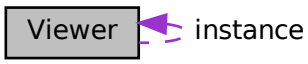
- src/graphics/Vector2D.hpp

3.17 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.

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

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.

3.17.1 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.

3.17.2 Member Function Documentation

3.17.2.1 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  }
```

3.17.2.2 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 }
```

3.17.2.3 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); }
```

3.17.2.4 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 }
```

3.17.2.5 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 }
```

3.17.2.6 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 }
```

3.17.2.7 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 }
```

3.17.3 Member Data Documentation

3.17.3.1 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