

raytracer

0.1.0

Generated by Doxygen 1.9.1



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 rtr::ALight Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Member Function Documentation	8
4.1.2.1 setType()	8
4.2 rtr::AMaterial Class Reference	8
4.2.1 Detailed Description	9
4.2.2 Member Function Documentation	9
4.2.2.1 setReflectivity()	9
4.2.2.2 setTransparency()	9
4.3 rtr::Ambient Class Reference	10
4.3.1 Member Function Documentation	10
4.3.1.1 getPluginName()	10
4.3.1.2 LightColor()	11
4.4 rtr::ARenderer Class Reference	11
4.4.1 Detailed Description	12
4.4.2 Member Function Documentation	12
4.4.2.1 getBackgroundColor()	12
4.4.2.2 getName()	12
4.4.2.3 getPixels()	13
4.4.2.4 getResolution()	13
4.4.2.5 getType()	13
4.4.2.6 setName()	13
4.4.2.7 setPixels()	14
4.4.2.8 setType()	14
4.5 rtr::AShape Class Reference	14
4.5.1 Detailed Description	15
4.5.2 Member Function Documentation	15
4.5.2.1 getDistance()	15
4.5.2.2 getHeight()	16
4.5.2.3 getMaterial()	16
4.5.2.4 getNormal()	16
4.5.2.5 getPosition()	17
4.5.2.6 getRadius()	17

4.5.2.7	<a href="#">getRotation()</a>	17
4.5.2.8	<a href="#">getType()</a>	17
4.5.2.9	<a href="#">setHeight()</a>	17
4.5.2.10	<a href="#">setMaterial()</a>	18
4.5.2.11	<a href="#">setRadius()</a>	18
4.5.2.12	<a href="#">setType()</a>	18
4.6	<a href="#">rtr::Camera Class Reference</a>	19
4.6.1	<a href="#">Detailed Description</a>	19
4.6.2	<a href="#">Member Function Documentation</a>	19
4.6.2.1	<a href="#">getFov()</a>	19
4.6.2.2	<a href="#">ray()</a>	19
4.6.2.3	<a href="#">setFov()</a>	20
4.7	<a href="#">rtr::Color Class Reference</a>	20
4.7.1	<a href="#">Detailed Description</a>	21
4.7.2	<a href="#">Member Function Documentation</a>	21
4.7.2.1	<a href="#">operator*()</a> [1/2]	21
4.7.2.2	<a href="#">operator*()</a> [2/2]	22
4.7.2.3	<a href="#">operator*=( )</a>	22
4.7.2.4	<a href="#">operator+( )</a>	22
4.7.2.5	<a href="#">operator+=( )</a>	24
4.7.2.6	<a href="#">setColor()</a> [1/2]	24
4.7.2.7	<a href="#">setColor()</a> [2/2]	24
4.8	<a href="#">color_s Struct Reference</a>	25
4.8.1	<a href="#">Detailed Description</a>	25
4.9	<a href="#">rtr::CompositeMaterial Class Reference</a>	25
4.9.1	<a href="#">Detailed Description</a>	26
4.9.2	<a href="#">Member Function Documentation</a>	26
4.9.2.1	<a href="#">addMaterial()</a>	26
4.9.2.2	<a href="#">getPluginName()</a>	26
4.10	<a href="#">rtr::Core Class Reference</a>	27
4.10.1	<a href="#">Detailed Description</a>	27
4.10.2	<a href="#">Member Function Documentation</a>	27
4.10.2.1	<a href="#">runRayTracer()</a>	27
4.11	<a href="#">rtr::Core::CoreException Class Reference</a>	27
4.11.1	<a href="#">Detailed Description</a>	28
4.12	<a href="#">rtr::Directional Class Reference</a>	28
4.12.1	<a href="#">Member Function Documentation</a>	29
4.12.1.1	<a href="#">getPluginName()</a>	29
4.12.1.2	<a href="#">LightColor()</a>	29
4.13	<a href="#">rtr::ILight Class Reference</a>	29
4.13.1	<a href="#">Detailed Description</a>	30
4.13.2	<a href="#">Member Function Documentation</a>	30

4.13.2.1 LightColor()	30
4.13.2.2 setType()	31
4.14 rtr::IMaterial Class Reference	31
4.14.1 Detailed Description	32
4.14.2 Member Function Documentation	32
4.14.2.1 setReflectivity()	32
4.14.2.2 setTransparency()	32
4.15 rtr::IPlugin Class Reference	33
4.15.1 Detailed Description	33
4.15.2 Member Function Documentation	33
4.15.2.1 getPluginName()	34
4.16 rtr::IRenderer Class Reference	34
4.16.1 Detailed Description	35
4.16.2 Member Function Documentation	35
4.16.2.1 getBackgroundColor()	35
4.16.2.2 getName()	35
4.16.2.3 getPixels()	35
4.16.2.4 getResolution()	36
4.16.2.5 getType()	36
4.16.2.6 render()	36
4.16.2.7 setName()	36
4.16.2.8 setPixels()	37
4.16.2.9 setType()	37
4.17 rtr::IShape Class Reference	37
4.17.1 Detailed Description	38
4.17.2 Member Function Documentation	38
4.17.2.1 getDistance()	39
4.17.2.2 getHeight()	39
4.17.2.3 getMaterial()	39
4.17.2.4 getNormal()	40
4.17.2.5 getPosition()	40
4.17.2.6 getRadius()	40
4.17.2.7 getRotation()	40
4.17.2.8 getType()	41
4.17.2.9 hits()	41
4.17.2.10 setHeight()	41
4.17.2.11 setMaterial()	42
4.17.2.12 setRadius()	42
4.17.2.13 setType()	42
4.18 rtr::LightFactory Class Reference	43
4.18.1 Detailed Description	43
4.18.2 Member Function Documentation	43

4.18.2.1 createLight() [1/2]	43
4.18.2.2 createLight() [2/2]	43
4.19 rtr::MaterialFactory Class Reference	44
4.19.1 Detailed Description	44
4.19.2 Member Function Documentation	44
4.19.2.1 createMaterial()	44
4.20 rtr::Parser Class Reference	45
4.20.1 Detailed Description	46
4.20.2 Member Function Documentation	46
4.20.2.1 convertInt()	46
4.20.2.2 getVector()	46
4.20.2.3 parseArgs()	47
4.20.2.4 parseCamera()	47
4.20.2.5 parseFile()	47
4.20.2.6 parseLights()	48
4.20.2.7 parseLightType()	48
4.20.2.8 parseMaterial()	48
4.20.2.9 parseRenderer()	49
4.20.2.10 parseShapes()	49
4.20.2.11 parseShapeType()	49
4.21 rtr::Parser::ParserException Class Reference	50
4.21.1 Detailed Description	50
4.22 rtr::PluginLoader Class Reference	51
4.22.1 Detailed Description	51
4.22.2 Member Function Documentation	51
4.22.2.1 getInstance()	51
4.22.2.2 getPlugin()	51
4.23 rtr::Point Class Reference	52
4.23.1 Member Function Documentation	52
4.23.1.1 getPluginName()	52
4.23.1.2 LightColor()	53
4.24 rtr::PPM Class Reference	54
4.24.1 Member Function Documentation	55
4.24.1.1 getPluginName()	55
4.24.1.2 render()	55
4.25 ray_hit_s Struct Reference	55
4.25.1 Detailed Description	56
4.26 rtr::RayHit Class Reference	56
4.26.1 Detailed Description	56
4.26.2 Member Function Documentation	56
4.26.2.1 setDistance()	56
4.26.2.2 setNormal()	57

4.26.2.3 setPoint()	57
4.26.2.4 setRayHit() [1/2]	57
4.26.2.5 setRayHit() [2/2]	58
4.27 rtr::Reflective Class Reference	58
4.27.1 Member Function Documentation	58
4.27.1.1 getPluginName()	59
4.28 rtr::RendererFactory Class Reference	59
4.28.1 Detailed Description	59
4.28.2 Member Function Documentation	59
4.28.2.1 createRenderer()	59
4.29 rtr::Resolution Class Reference	60
4.29.1 Detailed Description	60
4.29.2 Member Function Documentation	61
4.29.2.1 getHeight()	61
4.29.2.2 getValue()	61
4.29.2.3 getWidth()	61
4.29.2.4 setHeight()	61
4.29.2.5 setResolution() [1/2]	62
4.29.2.6 setResolution() [2/2]	62
4.29.2.7 setWidth()	62
4.30 resolution_s Struct Reference	63
4.30.1 Detailed Description	63
4.31 rtr::RunTimeException Class Reference	63
4.32 rtr::Scene Class Reference	63
4.32.1 Detailed Description	64
4.32.2 Member Function Documentation	64
4.32.2.1 addLight()	64
4.32.2.2 addShape()	64
4.32.2.3 getLights()	65
4.32.2.4 getRenderer()	65
4.32.2.5 getShapes()	65
4.32.2.6 setCamera()	65
4.32.2.7 setRenderer()	66
4.33 rtr::SFML Class Reference	66
4.33.1 Member Function Documentation	66
4.33.1.1 getPluginName()	67
4.33.1.2 render()	67
4.34 rtr::ShapeFactory Class Reference	67
4.34.1 Detailed Description	68
4.34.2 Member Function Documentation	68
4.34.2.1 createShape() [1/3]	68
4.34.2.2 createShape() [2/3]	68

4.34.2.3 createShape() [3/3]	69
4.35 rtr::Transparent Class Reference	69
4.35.1 Member Function Documentation	70
4.35.1.1 getPluginName()	70
4.36 rtr::Vector Class Reference	70
4.36.1 Member Function Documentation	71
4.36.1.1 cross()	71
4.36.1.2 dot()	71
4.36.1.3 getValue()	72
4.36.1.4 getX()	72
4.36.1.5 getY()	72
4.36.1.6 getZ()	72
4.36.1.7 length()	73
4.36.1.8 normalize()	73
4.36.1.9 operator*() [1/2]	73
4.36.1.10 operator*() [2/2]	73
4.36.1.11 operator+() [1/2]	74
4.36.1.12 operator+() [2/2]	74
4.36.1.13 operator-()	74
4.36.1.14 operator/()	75
4.36.1.15 setVector() [1/2]	75
4.36.1.16 setVector() [2/2]	75
4.36.1.17 setX()	76
4.36.1.18 setY()	76
4.36.1.19 setZ()	76
4.37 vector_s Struct Reference	77
4.37.1 Detailed Description	77
<b>5 File Documentation</b>	<b>79</b>
5.1 App/include/RayTracer/Abstraction/ALight.hpp File Reference	79
5.2 App/include/RayTracer/Abstraction/AMaterial.hpp File Reference	79
5.3 App/include/RayTracer/Abstraction/ARenderer.hpp File Reference	79
5.4 App/include/RayTracer/Abstraction/AShape.hpp File Reference	80
5.5 App/include/RayTracer/Abstraction/ILight.hpp File Reference	80
5.6 App/include/RayTracer/Abstraction/IMaterial.hpp File Reference	80
5.7 App/include/RayTracer/Abstraction/IPlugin.hpp File Reference	80
5.8 App/include/RayTracer/Abstraction/IRenderer.hpp File Reference	81
5.9 App/include/RayTracer/Abstraction/IShape.hpp File Reference	81
5.10 App/include/RayTracer/Composite/Material.hpp File Reference	81
5.11 App/include/RayTracer/Factory/Material.hpp File Reference	82
5.12 App/include/RayTracer/Constants.hpp File Reference	82
5.13 App/include/RayTracer/Core.hpp File Reference	82



---

5.14 App/include/RayTracer/Factory/Light.hpp File Reference . . . . .	82
5.15 App/include/RayTracer/Factory/Renderer.hpp File Reference . . . . .	83
5.16 App/include/RayTracer/Factory/Shape.hpp File Reference . . . . .	83
5.17 App/include/RayTracer/Loader/Plugin.hpp File Reference . . . . .	83
5.18 App/include/RayTracer/Parser.hpp File Reference . . . . .	83
5.19 App/include/RayTracer/Scene/Camera.hpp File Reference . . . . .	84
5.20 App/include/RayTracer/Scene/Scene.hpp File Reference . . . . .	84
5.21 App/include/RayTracer/Utils/Color.hpp File Reference . . . . .	84
5.22 App/include/RayTracer/Utils/RayHit.hpp File Reference . . . . .	85
5.23 App/include/RayTracer/Utils/Resolution.hpp File Reference . . . . .	85
5.24 App/include/RayTracer/Utils/Vector.hpp File Reference . . . . .	86
<b>Index</b>	<b>87</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

rtr::Camera . . . . .	19
rtr::Color . . . . .	20
color_s . . . . .	25
rtr::Core . . . . .	27
std::exception	
rtr::Core::CoreException . . . . .	27
rtr::Parser::ParserException . . . . .	50
rtr::RunTimeException . . . . .	63
rtr::IPlugin . . . . .	33
rtr::ILight . . . . .	29
rtr::ALight . . . . .	7
rtr::Ambient . . . . .	10
rtr::Directional . . . . .	28
rtr::Point . . . . .	52
rtr::IMaterial . . . . .	31
rtr::AMaterial . . . . .	8
rtr::CompositeMaterial . . . . .	25
rtr::Reflective . . . . .	58
rtr::Transparent . . . . .	69
rtr::IRenderer . . . . .	34
rtr::ARenderer . . . . .	11
rtr::PPM . . . . .	54
rtr::SFML . . . . .	66
rtr::IShape . . . . .	37
rtr::AShape . . . . .	14
rtr::LightFactory . . . . .	43
rtr::MaterialFactory . . . . .	44
rtr::Parser . . . . .	45
rtr::PluginLoader . . . . .	51
ray_hit_s . . . . .	55
rtr::RayHit . . . . .	56
rtr::RendererFactory . . . . .	59
rtr::Resolution . . . . .	60
resolution_s . . . . .	63
rtr::Scene . . . . .	63
rtr::ShapeFactory . . . . .	67
rtr::Vector . . . . .	70
vector_s . . . . .	77



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">rtr::ALight</a>	An abstract class for lights . . . . .	7
<a href="#">rtr::AMaterial</a>	An abstract class for materials, based on the interface <a href="#">IMaterials</a> . . . . .	8
<a href="#">rtr::Ambient</a>	. . . . .	10
<a href="#">rtr::ARenderer</a>	An abstract class for renderers, based on the interface <a href="#">IRenderer</a> . . . . .	11
<a href="#">rtr::AShape</a>	An abstract class for shapes, based on the interface <a href="#">IShape</a> . . . . .	14
<a href="#">rtr::Camera</a>	A class to handle the camera . . . . .	19
<a href="#">rtr::Color</a>	Class representing RGB colors . . . . .	20
<a href="#">color_s</a>	A struct representing an RGB color . . . . .	25
<a href="#">rtr::CompositeMaterial</a>	A class to create a composite material . . . . .	25
<a href="#">rtr::Core</a>	A class representing the core functionality of the ray tracer . . . . .	27
<a href="#">rtr::Core::CoreException</a>	An exception class for core errors . . . . .	27
<a href="#">rtr::Directional</a>	. . . . .	28
<a href="#">rtr::ILight</a>	An interface for lights . . . . .	29
<a href="#">rtr::IMaterial</a>	An interface for materials . . . . .	31
<a href="#">rtr::IPlugin</a>	An interface for plugins . . . . .	33
<a href="#">rtr::IRenderer</a>	An interface for renderers . . . . .	34
<a href="#">rtr::IShape</a>	An interface used to get the shape's parameters based on the configuration file . . . . .	37
<a href="#">rtr::LightFactory</a>	A factory class for the lights . . . . .	43
<a href="#">rtr::MaterialFactory</a>	A factory class for the materials of the shapes . . . . .	44

<a href="#">rtr::Parser</a>	Class dedicated to the parsing of configuration files and command-line arguments . . . . .	45
<a href="#">rtr::Parser::ParserException</a>	Exception class for errors in the parsers . . . . .	50
<a href="#">rtr::PluginLoader</a>	A class to load the plugins . . . . .	51
<a href="#">rtr::Point</a>	. . . . .	52
<a href="#">rtr::PPM</a>	. . . . .	54
<a href="#">ray_hit_s</a>	A struct representing a ray hit in 3D space . . . . .	55
<a href="#">rtr::RayHit</a>	A class representing a ray hit in 3D space . . . . .	56
<a href="#">rtr::Reflective</a>	. . . . .	58
<a href="#">rtr::RendererFactory</a>	A factory class to create the renderers . . . . .	59
<a href="#">rtr::Resolution</a>	Class representing the resolution of an image . . . . .	60
<a href="#">resolution_s</a>	A struct representing the resolution of an image . . . . .	63
<a href="#">rtr::RunTimeException</a>	. . . . .	63
<a href="#">rtr::Scene</a>	A class to represent the scene . . . . .	63
<a href="#">rtr::SFML</a>	. . . . .	66
<a href="#">rtr::ShapeFactory</a>	A factory class for the shapes . . . . .	67
<a href="#">rtr::Transparent</a>	. . . . .	69
<a href="#">rtr::Vector</a>	. . . . .	70
<a href="#">vector_s</a>	A struct representing a 3D vector . . . . .	77

## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

App/include/RayTracer/Constants.hpp	82
App/include/RayTracer/Core.hpp	82
App/include/RayTracer/Parser.hpp	83
App/include/RayTracer/Abstraction/ALight.hpp	79
App/include/RayTracer/Abstraction/AMaterial.hpp	79
App/include/RayTracer/Abstraction/ARenderer.hpp	79
App/include/RayTracer/Abstraction/AShape.hpp	80
App/include/RayTracer/Abstraction/ILight.hpp	80
App/include/RayTracer/Abstraction/IMaterial.hpp	80
App/include/RayTracer/Abstraction/IPlugin.hpp	80
App/include/RayTracer/Abstraction/IRenderer.hpp	81
App/include/RayTracer/Abstraction/IShape.hpp	81
App/include/RayTracer/Composite/Material.hpp	81
App/include/RayTracer/Exception/RunTime.hpp	??
App/include/RayTracer/Factory/Light.hpp	82
App/include/RayTracer/Factory/Material.hpp	82
App/include/RayTracer/Factory/Renderer.hpp	83
App/include/RayTracer/Factory/Shape.hpp	83
App/include/RayTracer/Loader/Plugin.hpp	83
App/include/RayTracer/Scene/Camera.hpp	84
App/include/RayTracer/Scene/Scene.hpp	84
App/include/RayTracer/Utils/Color.hpp	84
App/include/RayTracer/Utils/RayHit.hpp	85
App/include/RayTracer/Utils/Resolution.hpp	85
App/include/RayTracer/Utils/Vector.hpp	86
App/plugins/Light/Ambient/include/RayTracer/Ambient.hpp	??
App/plugins/Light/Directional/include/RayTracer/Directional.hpp	??
App/plugins/Light/Point/include/RayTracer/Point.hpp	??
App/plugins/Material/Reflective/include/RayTracer/Reflective.hpp	??
App/plugins/Material/Transparent/include/RayTracer/Transparent.hpp	??
App/plugins/Renderer/PPM/include/RayTracer/PPM.hpp	??
App/plugins/Renderer/SFML/include/RayTracer/SFML.hpp	??





## Chapter 4

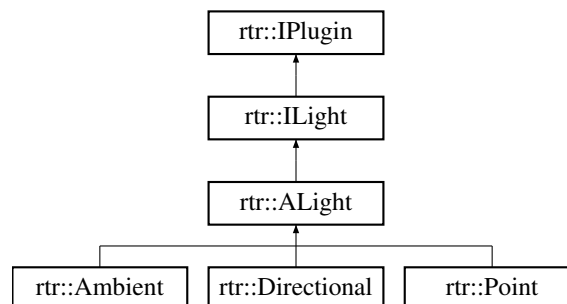
# Class Documentation

### 4.1 rtr::ALight Class Reference

An abstract class for lights.

```
#include <ALight.hpp>
```

Inheritance diagram for rtr::ALight:



#### Public Member Functions

- void [setType](#) (const LightType &type) override  
*Sets the type of the light (directional, ambient or point).*
- void [setIntensity](#) (const float &intensity) override  
*Sets the intensity of the light, based on the configuration file.*
- const LightType & [getType](#) () const override  
*Gets the type of the light based on the configuration file.*
- [Vector](#) & [getPosition](#) () override  
*Gets the position of the light based on the configuration file.*
- [Vector](#) & [getDirection](#) () override  
*Gets the direction of the light based on the configuration file.*
- [Color](#) & [getColor](#) () override  
*Gets the color of the light based on the configuration file.*
- float & [getIntensity](#) () override  
*Gets the intensity of the light based on the configuration file.*

### 4.1.1 Detailed Description

An abstract class for lights.

### 4.1.2 Member Function Documentation

#### 4.1.2.1 setType()

```
void rtr::ALight::setType (
    const LightType & type ) [inline], [override], [virtual]
```

Sets the type of the light (directional, ambient or point).

#### Parameters

<i>type</i>	The type of the light (defined in the enum class LighType).
-------------	---

Implements [rtr::ILight](#).

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

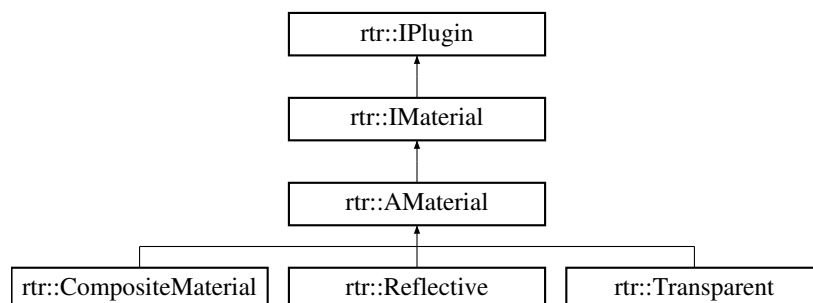
- [App/include/RayTracer/Abstraction/ALight.hpp](#)

## 4.2 rtr::AMaterial Class Reference

An abstract class for materials, based on the interface IMaterials.

```
#include <AMaterial.hpp>
```

Inheritance diagram for rtr::AMaterial:



## Public Member Functions

- void [setType](#) (const MaterialType &type) override  
*Sets the type of the material.*
- void [setReflectivity](#) (const float &reflectivity) override  
*Sets the color of the material.*
- void [setTransparency](#) (const float &transparency) override  
*Sets the transparency of the material.*
- const MaterialType & [getType](#) () const override  
*Gets the type of the material based on the configuration file.*
- [Color](#) & [getColor](#) () override  
*Gets the color of the material based on the configuration file.*
- const float & [getReflectivity](#) () const override  
*Gets the reflectiveness of the material based on the configuration file.*
- const float & [getTransparency](#) () const override  
*Gets the transparency of the material based on the configuration file.*

### 4.2.1 Detailed Description

An abstract class for materials, based on the interface IMaterials.

### 4.2.2 Member Function Documentation

#### 4.2.2.1 setReflectivity()

```
void rtr::AMaterial::setReflectivity (
    const float & reflectivity ) [inline], [override], [virtual]
```

Sets the color of the material.

Parameters

<i>reflectivity</i>	The reflectiveness of the material, based on the configuration file.
---------------------	--

Implements [rtr::IMaterial](#).

#### 4.2.2.2 setTransparency()

```
void rtr::AMaterial::setTransparency (
    const float & transparency ) [inline], [override], [virtual]
```

Sets the transparency of the material.

## Parameters

<i>transparency</i>	The transparency of the material, based on the configuration file.
---------------------	--

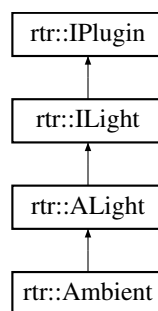
Implements [rtr::IMaterial](#).

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

- [App/include/RayTracer/Abstraction/AMaterial.hpp](#)

## 4.3 rtr::Ambient Class Reference

Inheritance diagram for rtr::Ambient:



### Public Member Functions

- [Color LightColor](#) (const [Vector](#) &normal, const [Color](#) &col) override  
*Creates light effects based on the light type.*
- [std::string getPluginName](#) () const override  
*Gets the name of the plugin.*
- [Vector &getDirection](#) () override  
*Gets the direction of the light based on the configuration file.*

### 4.3.1 Member Function Documentation

#### 4.3.1.1 getPluginName()

```
std::string rtr::Ambient::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

#### Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

## 4.3.1.2 LightColor()

```
Color rtr::Ambient::LightColor (
    const Vector & normal,
    const Color & col ) [override], [virtual]
```

Creates light effects based on the light type.

## Parameters

<i>normal</i>	The normal of the shape.
<i>col</i>	The current color of the shape.

## Returns

The new color of the shape with the light effects.

Implements [rtr::ILight](#).

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

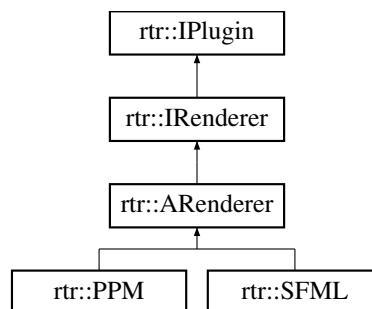
- App/plugins/Light/Ambient/include/RayTracer/Ambient.hpp

## 4.4 rtr::ARenderer Class Reference

An abstract class for renderers, based on the interface [IRenderer](#).

```
#include <ARenderer.hpp>
```

Inheritance diagram for rtr::ARenderer:



## Public Member Functions

- void [setType](#) (const `RendererType` &`rendererType`) override  
*Sets the type of the renderer.*
- void [setName](#) (const `std::string` &`name`) override  
*Sets the name of the renderer.*
- void [setPixels](#) (const `std::vector`< `std::vector`< [rtr::Color](#) >> &`pixels`) override  
*Sets the pixels of the renderer.*
- const `RendererType` & [getType](#) () const override  
*Gets the type of the renderer based on the configuration file.*
- [Resolution](#) & [getResolution](#) () override  
*Gets the resolution of the renderer based on the configuration file.*
- [Color](#) & [getBackgroundColor](#) () override  
*Gets the background color of the renderer based on the configuration file.*
- const `std::string` & [getName](#) () const override  
*Gets the name of the renderer based on the configuration file.*
- `std::vector`< `std::vector`< [rtr::Color](#) >> & [getPixels](#) () override  
*Gets the pixels of the renderer based on the configuration file.*

### 4.4.1 Detailed Description

An abstract class for renderers, based on the interface [IRenderer](#).

### 4.4.2 Member Function Documentation

#### 4.4.2.1 [getBackgroundColor\(\)](#)

```
Color& rtr::ARenderer::getBackgroundColor ( ) [inline], [override], [virtual]
```

Gets the background color of the renderer based on the configuration file.

##### Returns

The background color of the renderer.

Implements [rtr::IRenderer](#).

#### 4.4.2.2 [getName\(\)](#)

```
const std::string& rtr::ARenderer::getName ( ) const [inline], [override], [virtual]
```

Gets the name of the renderer based on the configuration file.

##### Returns

A string of the renderer's name.

Implements [rtr::IRenderer](#).

#### 4.4.2.3 getPixels()

```
std::vector<std::vector<rtr::Color> >& rtr::ARenderer::getPixels ( ) [inline], [override], [virtual]
```

Gets the pixels of the renderer based on the configuration file.

##### Returns

Each pixels of the image.

Implements [rtr::IRenderer](#).

#### 4.4.2.4 getResolution()

```
Resolution& rtr::ARenderer::getResolution ( ) [inline], [override], [virtual]
```

Gets the resolution of the renderer based on the configuration file.

##### Returns

The resolution of the renderer using the [Resolution](#) class.

Implements [rtr::IRenderer](#).

#### 4.4.2.5 getType()

```
const RendererType& rtr::ARenderer::getType ( ) const [inline], [override], [virtual]
```

Gets the type of the renderer based on the configuration file.

##### Returns

The type of the renderer, using the enum class `RendererType`.

Implements [rtr::IRenderer](#).

#### 4.4.2.6 setName()

```
void rtr::ARenderer::setName (
    const std::string & name ) [inline], [override], [virtual]
```

Sets the name of the renderer.

## Parameters

<i>name</i>	The name of the renderer, based on the configuration file.
-------------	--

Implements [rtr::IRenderer](#).

**4.4.2.7 setPixels()**

```
void rtr::ARenderer::setPixels (
    const std::vector< std::vector< rtr::Color >> & pixels ) [inline], [override],
[virtual]
```

Sets the pixels of the renderer.

## Parameters

<i>pixels</i>	The pixels of the renderer.
---------------	-----------------------------

Implements [rtr::IRenderer](#).

**4.4.2.8 setType()**

```
void rtr::ARenderer::setType (
    const RendererType & rendererType ) [inline], [override], [virtual]
```

Sets the type of the renderer.

## Parameters

<i>rendererType</i>	The type of the renderer (defined in the RendererType enum class).
---------------------	--

Implements [rtr::IRenderer](#).

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

- App/include/RayTracer/Abstraction/[ARenderer.hpp](#)

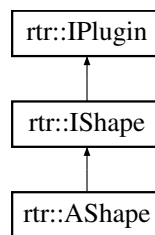
**4.5 rtr::AShape Class Reference**

An abstract class for shapes, based on the interface [IShape](#).

```
#include <AShape.hpp>
```



Inheritance diagram for rtr::AShape:



## Public Member Functions

- void [setType](#) (const ShapeType &type) override  
*Sets the type of the shape (sphere, plane, cone...).*
- void [setRadius](#) (const double &radius) override  
*Sets the radius of the shape.*
- void [setHeight](#) (const double &height) override  
*Sets the height of the shape.*
- void [setMaterial](#) (std::unique\_ptr< [AMaterial](#) > material) override  
*Sets the material of the shape.*
- const ShapeType & [getType](#) () const override  
*Gets the type of the shape.*
- [AMaterial](#) & [getMaterial](#) () override  
*Gets the material of the shape.*
- [Vector](#) & [getPosition](#) () override  
*Gets the position of the shape.*
- [Vector](#) & [getNormal](#) () override  
*Gets the normal of the shape.*
- [Vector](#) & [getRotation](#) () override  
*Gets the rotation of the shape, used to create the shape.*
- const double & [getRadius](#) () const override  
*Gets the radius of the shape.*
- const double & [getHeight](#) () const override  
*Gets the height of the shape.*
- [Vector](#) [getDistance](#) (const [Vector](#) &point) override  
*Gets the distance between the shape and a point.*

### 4.5.1 Detailed Description

An abstract class for shapes, based on the interface [IShape](#).

### 4.5.2 Member Function Documentation

#### 4.5.2.1 getDistance()

```

Vector rtr::AShape::getDistance (
    const Vector & point ) [inline], [override], [virtual]
  
```

Gets the distance between the shape and a point.

**Parameters**

<i>point</i>	The point to check the distance with.
--------------	---------------------------------------

**Returns**

The distance between the shape and the point.

Implements [rtr::IShape](#).

**4.5.2.2 getHeight()**

```
const double& rtr::AShape::getHeight ( ) const [inline], [override], [virtual]
```

Gets the height of the shape.

**Returns**

The height of the shape as a double.

Implements [rtr::IShape](#).

**4.5.2.3 getMaterial()**

```
AMaterial& rtr::AShape::getMaterial ( ) [inline], [override], [virtual]
```

Gets the material of the shape.

**Returns**

The material of the shape, using the [AMaterial](#) class.

Implements [rtr::IShape](#).

**4.5.2.4 getNormal()**

```
Vector& rtr::AShape::getNormal ( ) [inline], [override], [virtual]
```

Gets the normal of the shape.

**Returns**

The normal of the shape, using the [Vector](#) class.

Implements [rtr::IShape](#).

#### 4.5.2.5 getPosition()

```
Vector& rtr::AShape::getPosition ( ) [inline], [override], [virtual]
```

Gets the position of the shape.

##### Returns

The position of the shape, using the [Vector](#) class.

Implements [rtr::IShape](#).

#### 4.5.2.6 getRadius()

```
const double& rtr::AShape::getRadius ( ) const [inline], [override], [virtual]
```

Gets the radius of the shape.

##### Returns

The radius of the shape as a double, used to check the size of the shape.

Implements [rtr::IShape](#).

#### 4.5.2.7 getRotation()

```
Vector& rtr::AShape::getRotation ( ) [inline], [override], [virtual]
```

Gets the rotation of the shape, used to create the shape.

##### Returns

The rotation of the shape, using the [Vector](#) class.

Implements [rtr::IShape](#).

#### 4.5.2.8 getType()

```
const ShapeType& rtr::AShape::getType ( ) const [inline], [override], [virtual]
```

Gets the type of the shape.

##### Returns

The type of the shape, using the ShapeType enum class.

Implements [rtr::IShape](#).

#### 4.5.2.9 setHeight()

```
void rtr::AShape::setHeight (
    const double & height ) [inline], [override], [virtual]
```

Sets the height of the shape.

**Parameters**

<i>height</i>	The height of the shape.
---------------	--------------------------

Implements [rtr::IShape](#).

**4.5.2.10 setMaterial()**

```
void rtr::AShape::setMaterial (
    std::unique_ptr< AMaterial > material ) [inline], [override], [virtual]
```

Sets the material of the shape.

**Parameters**

<i>material</i>	The material of the shape (transparency, reflectivity...).
-----------------	--

Implements [rtr::IShape](#).

**4.5.2.11 setRadius()**

```
void rtr::AShape::setRadius (
    const double & radius ) [inline], [override], [virtual]
```

Sets the radius of the shape.

**Parameters**

<i>radius</i>	The radius of the shape.
---------------	--------------------------

Implements [rtr::IShape](#).

**4.5.2.12 setType()**

```
void rtr::AShape::setType (
    const ShapeType & type ) [inline], [override], [virtual]
```

Sets the type of the shape (sphere, plane, cone...).

**Parameters**

<i>type</i>	The type of the shape.
-------------	------------------------

Implements [rtr::IShape](#).

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

- App/include/RayTracer/Abstraction/[AShape.hpp](#)

## 4.6 rtr::Camera Class Reference

A class to handle the camera.

```
#include <Camera.hpp>
```

### Public Member Functions

- **Camera** (uint16\_t fov, const [Vector](#) &origin, const [Vector](#) &direction)
- void [setFov](#) (const uint16\_t fov)  
*Sets the camera's field of view.*
- uint16\_t [getFov](#) () const  
*Gets the camera's field of view.*
- const [Vector](#) & [getOrigin](#) () const  
*Gets the camera's origin.*
- const [Vector](#) & [getDirection](#) () const  
*Gets the camera's direction.*
- const [Vector](#) & [getUp](#) () const  
*Gets the camera's up vector.*
- std::pair< [Vector](#), [Vector](#) > [ray](#) (const double u, const double v) const  
*The camera's ray, used to check if the ray intersects with an object.*

### 4.6.1 Detailed Description

A class to handle the camera.

### 4.6.2 Member Function Documentation

#### 4.6.2.1 getFov()

```
uint16_t rtr::Camera::getFov ( ) const [inline]
```

Gets the camera's field of view.

#### Returns

The field of view, as a uint16, or a unsigned short.

#### 4.6.2.2 ray()

```
std::pair<Vector, Vector> rtr::Camera::ray (
    const double u,
    const double v ) const [inline]
```

The camera's ray, used to check if the ray intersects with an object.

**Parameters**

<i>u</i>	defines the horizontal position of the ray.
<i>v</i>	defines the vertical position of the ray.

**Returns**

A pair of vectors that defines the origin of the ray and its normal.

**4.6.2.3 setFov()**

```
void rtr::Camera::setFov (
    const uint16_t fov ) [inline]
```

Sets the camera's field of view.

**Parameters**

<i>fov</i>	The field of view, as a uint16, or a unsigned short.
------------	--

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

- App/include/RayTracer/Scene/[Camera.hpp](#)

**4.7 rtr::Color Class Reference**

Class representing RGB colors.

```
#include <Color.hpp>
```

**Public Member Functions**

- **Color** (const uint8\_t r, const uint8\_t g, const uint8\_t b)
- **Color** (const color\_t &color)
- void **setColor** (const uint8\_t r, const uint8\_t g, const uint8\_t b)  
*Sets the color components.*
- void **setColor** (const color\_t &color)  
*Sets the color components.*
- void **setR** (const uint8\_t r)
- void **setG** (const uint8\_t g)
- void **setB** (const uint8\_t b)
- color\_t **getValue** () const
- uint8\_t **getR** () const
- uint8\_t **getG** () const
- uint8\_t **getB** () const

- **Color operator+** (const **Color** &other) const  
*Adds two colors.*
- **Color operator\*** (const double &scalar) const  
*Multiplies a color by a scalar.*
- **Color operator\*** (const **Color** &other) const  
*Multiplies two colors.*
- **Color operator+=** (const **Color** &other)  
*Adds a color to the current color.*
- **Color operator\*=** (const double &scalar)  
*Multiplies the current color by a scalar.*

## Static Public Member Functions

- static constexpr color\_t **getRed** ()
- static constexpr color\_t **getGreen** ()
- static constexpr color\_t **getBlue** ()
- static constexpr color\_t **getWhite** ()
- static constexpr color\_t **getBlack** ()
- static constexpr color\_t **getYellow** ()
- static constexpr color\_t **getMagenta** ()
- static constexpr color\_t **getCyan** ()
- static constexpr color\_t **getGray** ()
- static constexpr color\_t **getOrange** ()
- static constexpr color\_t **getBrown** ()
- static constexpr color\_t **getLightBlue** ()
- static constexpr color\_t **getLightGreen** ()
- static constexpr color\_t **getLightPink** ()
- static constexpr color\_t **getLightYellow** ()
- static constexpr color\_t **getLightGray** ()
- static constexpr color\_t **getDarkGray** ()
- static constexpr color\_t **getDarkRed** ()
- static constexpr color\_t **getDarkGreen** ()
- static constexpr color\_t **getDarkBlue** ()
- static constexpr color\_t **getDarkYellow** ()

### 4.7.1 Detailed Description

Class representing RGB colors.

### 4.7.2 Member Function Documentation

#### 4.7.2.1 operator\*() [1/2]

```
Color rtr::Color::operator* (
    const Color & other ) const [inline]
```

Multiplies two colors.

**Parameters**

<i>other</i>	The other color to multiply.
--------------	------------------------------

**Returns**

The product of the two colors.

**4.7.2.2 operator\*() [2/2]**

```
Color rtr::Color::operator* (
    const double & scalar ) const [inline]
```

Multiplies a color by a scalar.

**Parameters**

<i>scalar</i>	The scalar to multiply by.
---------------	----------------------------

**Returns**

The product of the color and the scalar.

**4.7.2.3 operator\*=( )**

```
Color rtr::Color::operator*= (
    const double & scalar ) [inline]
```

Multiplies the current color by a scalar.

**Parameters**

<i>scalar</i>	The scalar to multiply by.
---------------	----------------------------

**Returns**

A reference to the current color.

**4.7.2.4 operator+()**

```
Color rtr::Color::operator+ (
    const Color & other ) const [inline]
```



Adds two colors.

**Parameters**

<i>other</i>	The other color to add.
--------------	-------------------------

**Returns**

The sum of the two colors.

**4.7.2.5 operator+=()**

```
Color rtr::Color::operator+= (
    const Color & other ) [inline]
```

Adds a color to the current color.

**Parameters**

<i>other</i>	The other color to add.
--------------	-------------------------

**Returns**

A reference to the current color.

**4.7.2.6 setColor() [1/2]**

```
void rtr::Color::setColor (
    const color_t & color ) [inline]
```

Sets the color components.

**Parameters**

<i>color</i>	An RGB color.
--------------	---------------

**4.7.2.7 setColor() [2/2]**

```
void rtr::Color::setColor (
    const uint8_t r,
    const uint8_t g,
    const uint8_t b ) [inline]
```

Sets the color components.

## Parameters

<i>r</i>	Red color component.
<i>g</i>	Green color component.
<i>b</i>	Blue color component.

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

- App/include/RayTracer/Utils/[Color.hpp](#)

## 4.8 color\_s Struct Reference

A struct representing an RGB color.

```
#include <Color.hpp>
```

### 4.8.1 Detailed Description

A struct representing an RGB color.

Type alias for an RGB color component.

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

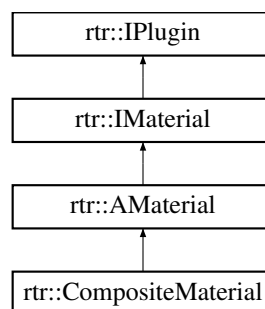
- App/include/RayTracer/Utils/[Color.hpp](#)

## 4.9 rtr::CompositeMaterial Class Reference

A class to create a composite material.

```
#include <Material.hpp>
```

Inheritance diagram for rtr::CompositeMaterial:



## Public Member Functions

- `std::string getPluginName ()` const override  
*Gets the name of the plugin.*
- `void addMaterial (std::unique_ptr< AMaterial > material)`  
*Adds a material to the composite material.*
- `void applyMaterial (Color *color)` override  
*Applies the material to the shape (transparency and reflectiveness).*

### 4.9.1 Detailed Description

A class to create a composite material.

### 4.9.2 Member Function Documentation

#### 4.9.2.1 `addMaterial()`

```
void rtr::CompositeMaterial::addMaterial (
    std::unique_ptr< AMaterial > material ) [inline]
```

Adds a material to the composite material.

##### Parameters

<i>material</i>	The material to add.
-----------------	----------------------

#### 4.9.2.2 `getPluginName()`

```
std::string rtr::CompositeMaterial::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

##### Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

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

- `App/include/RayTracer/Composite/Material.hpp`

## 4.10 rtr::Core Class Reference

A class representing the core functionality of the ray tracer.

```
#include <Core.hpp>
```

### Classes

- class [CoreException](#)  
*An exception class for core errors.*

### Static Public Member Functions

- static void [runRayTracer](#) ([Scene](#) &scene)  
*Runs the ray tracer with a given [Scene](#) object.*

#### 4.10.1 Detailed Description

A class representing the core functionality of the ray tracer.

#### 4.10.2 Member Function Documentation

##### 4.10.2.1 runRayTracer()

```
static void rtr::Core::runRayTracer (  
    Scene & scene ) [static]
```

Runs the ray tracer with a given [Scene](#) object.

##### Parameters

<i>scene</i>	The <a href="#">Scene</a> object used for ray tracing.
--------------	--

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

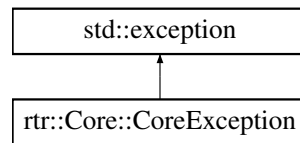
- App/include/RayTracer/[Core.hpp](#)

## 4.11 rtr::Core::CoreException Class Reference

An exception class for core errors.

```
#include <Core.hpp>
```

Inheritance diagram for `rtr::Core::CoreException`:



## Public Member Functions

- **CoreException** (std::string msg)
- **CoreException** (const [CoreException](#) &)=delete
- **CoreException** & **operator=** (const [CoreException](#) &)=delete
- **CoreException** (const [CoreException](#) &&)=delete
- **CoreException** & **operator=** (const [CoreException](#) &&)=delete
- const char \* **what** () const noexcept override

*Returns the error message.*

### 4.11.1 Detailed Description

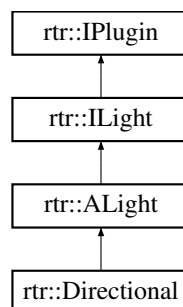
An exception class for core errors.

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

- App/include/RayTracer/[Core.hpp](#)

## 4.12 rtr::Directional Class Reference

Inheritance diagram for `rtr::Directional`:



## Public Member Functions

- **Color LightColor** (const [Vector](#) &normal, const [Color](#) &col) override
  - std::string **getPluginName** () const override
- Creates light effects based on the light type.*
- Gets the name of the plugin.*

## 4.12.1 Member Function Documentation

### 4.12.1.1 getPluginName()

```
std::string rtr::Directional::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

#### Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

### 4.12.1.2 LightColor()

```
Color rtr::Directional::LightColor (
    const Vector & normal,
    const Color & col ) [override], [virtual]
```

Creates light effects based on the light type.

#### Parameters

<i>normal</i>	The normal of the shape.
<i>col</i>	The current color of the shape.

#### Returns

The new color of the shape with the light effects.

Implements [rtr::ILight](#).

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

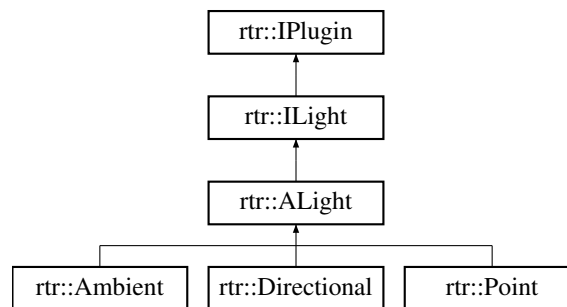
- App/plugins/Light/Directional/include/RayTracer/Directional.hpp

## 4.13 rtr::ILight Class Reference

An interface for lights.

```
#include <ILight.hpp>
```

Inheritance diagram for `rtr::ILight`:



## Public Member Functions

- virtual void `setType` (const `LightType` &type)=0  
*Sets the type of the light (directional, ambient or point).*
- virtual void `setIntensity` (const float &intensity)=0  
*Sets the intensity of the light, based on the configuration file.*
- virtual `Color` `LightColor` (const `Vector` &normal, const `Color` &col)=0  
*Creates light effects based on the light type.*
- virtual const `LightType` & `getType` () const =0  
*Gets the type of the light based on the configuration file.*
- virtual `Vector` & `getPosition` ()=0  
*Gets the position of the light based on the configuration file.*
- virtual `Vector` & `getDirection` ()=0  
*Gets the direction of the light based on the configuration file.*
- virtual `Color` & `getColor` ()=0  
*Gets the color of the light based on the configuration file.*
- virtual float & `getIntensity` ()=0  
*Gets the intensity of the light based on the configuration file.*

### 4.13.1 Detailed Description

An interface for lights.

### 4.13.2 Member Function Documentation

#### 4.13.2.1 `LightColor()`

```
virtual Color rtr::ILight::LightColor (
    const Vector & normal,
    const Color & col ) [pure virtual]
```

Creates light effects based on the light type.



## Parameters

<i>normal</i>	The normal of the shape.
<i>col</i>	The current color of the shape.

## Returns

The new color of the shape with the light effects.

Implemented in [rtr::Point](#), [rtr::Directional](#), and [rtr::Ambient](#).

## 4.13.2.2 setType()

```
virtual void rtr::ILight::setType (
    const LightType & type ) [pure virtual]
```

Sets the type of the light (directional, ambient or point).

## Parameters

<i>type</i>	The type of the light (defined in the enum class LighType).
-------------	---

Implemented in [rtr::ALight](#).

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

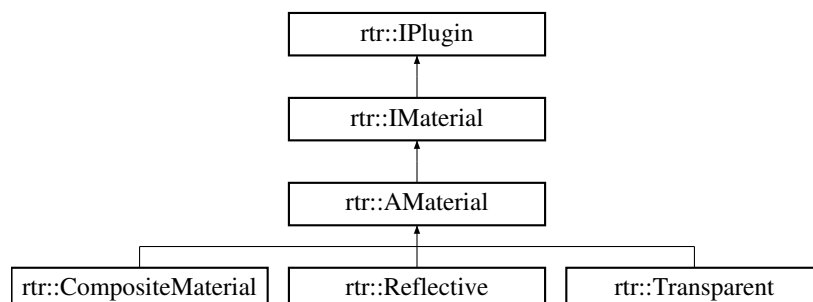
- [App/include/RayTracer/Abstraction/ILight.hpp](#)

## 4.14 rtr::IMaterial Class Reference

An interface for materials.

```
#include <IMaterial.hpp>
```

Inheritance diagram for rtr::IMaterial:



## Public Member Functions

- virtual void [applyMaterial](#) ([Color](#) \*color)=0  
*Applies the material to the shape (transparency and reflectiveness).*
- virtual void [setType](#) (const MaterialType &type)=0  
*Sets the type of the material.*
- virtual void [setReflectivity](#) (const float &reflectivity)=0  
*Sets the color of the material.*
- virtual void [setTransparency](#) (const float &transparency)=0  
*Sets the transparency of the material.*
- virtual const MaterialType & [getType](#) () const =0  
*Gets the type of the material based on the configuration file.*
- virtual [Color](#) & [getColor](#) ()=0  
*Gets the color of the material based on the configuration file.*
- virtual const float & [getReflectivity](#) () const =0  
*Gets the reflectiveness of the material based on the configuration file.*
- virtual const float & [getTransparency](#) () const =0  
*Gets the transparency of the material based on the configuration file.*

### 4.14.1 Detailed Description

An interface for materials.

### 4.14.2 Member Function Documentation

#### 4.14.2.1 setReflectivity()

```
virtual void rtr::IMaterial::setReflectivity (
    const float & reflectivity ) [pure virtual]
```

Sets the color of the material.

#### Parameters

<i>reflectivity</i>	The reflectiveness of the material, based on the configuration file.
---------------------	--

Implemented in [rtr::AMaterial](#).

#### 4.14.2.2 setTransparency()

```
virtual void rtr::IMaterial::setTransparency (
    const float & transparency ) [pure virtual]
```

Sets the transparency of the material.

## Parameters

<i>transparency</i>	The transparency of the material, based on the configuration file.
---------------------	--

Implemented in [rtr::AMaterial](#).

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

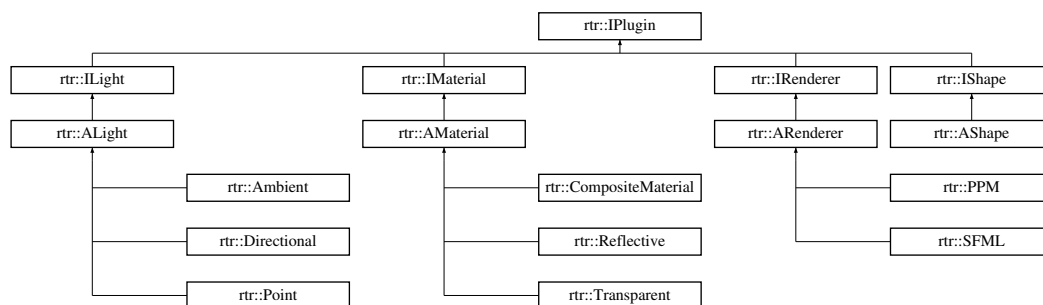
- App/include/RayTracer/Abstraction/[IMaterial.hpp](#)

## 4.15 rtr::IPlugin Class Reference

An interface for plugins.

```
#include <IPlugin.hpp>
```

Inheritance diagram for rtr::IPlugin:



### Public Member Functions

- virtual std::string [getPluginName](#) () const =0  
*Gets the name of the plugin.*

#### 4.15.1 Detailed Description

An interface for plugins.

#### 4.15.2 Member Function Documentation

#### 4.15.2.1 `getPluginName()`

```
virtual std::string rtr::IPlugin::getPluginName ( ) const [pure virtual]
```

Gets the name of the plugin.

##### Returns

A string of the plugin's name, defined as const expressions.

Implemented in [rtr::SFML](#), [rtr::PPM](#), [rtr::Transparent](#), [rtr::Reflective](#), [rtr::Point](#), [rtr::Directional](#), [rtr::Ambient](#), and [rtr::CompositeMaterial](#).

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

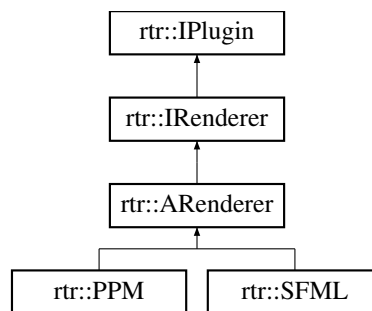
- [App/include/RayTracer/Abstraction/IPlugin.hpp](#)

## 4.16 `rtr::IRenderer` Class Reference

An interface for renderers.

```
#include <IRenderer.hpp>
```

Inheritance diagram for `rtr::IRenderer`:



### Public Member Functions

- virtual void [render](#) (const std::vector< std::unique\_ptr< [AShape](#) >> &shapes, const std::vector< std::unique\_ptr< [ALight](#) >> &lights, const [Camera](#) &camera)=0  
*Renders the scene based on the shapes, lights and camera.*
- virtual void [setType](#) (const `RendererType` &rendererType)=0  
*Sets the type of the renderer.*
- virtual void [setName](#) (const std::string &name)=0  
*Sets the name of the renderer.*
- virtual void [setPixels](#) (const std::vector< std::vector< [rtr::Color](#) >> &pixels)=0  
*Sets the pixels of the renderer.*
- virtual const `RendererType` & [getType](#) () const =0  
*Gets the type of the renderer based on the configuration file.*
- virtual const std::string & [getName](#) () const =0  
*Gets the name of the renderer based on the configuration file.*
- virtual [Resolution](#) & [getResolution](#) ()=0  
*Gets the resolution of the renderer based on the configuration file.*
- virtual [Color](#) & [getBackgroundColor](#) ()=0  
*Gets the background color of the renderer based on the configuration file.*
- virtual std::vector< std::vector< [rtr::Color](#) > > & [getPixels](#) ()=0  
*Gets the pixels of the renderer based on the configuration file.*

### 4.16.1 Detailed Description

An interface for renderers.

### 4.16.2 Member Function Documentation

#### 4.16.2.1 getBackgroundColor()

```
virtual Color& rtr::IRenderer::getBackgroundColor ( ) [pure virtual]
```

Gets the background color of the renderer based on the configuration file.

##### Returns

The background color of the renderer.

Implemented in [rtr::ARenderer](#).

#### 4.16.2.2 getName()

```
virtual const std::string& rtr::IRenderer::getName ( ) const [pure virtual]
```

Gets the name of the renderer based on the configuration file.

##### Returns

A string of the renderer's name.

Implemented in [rtr::ARenderer](#).

#### 4.16.2.3 getPixels()

```
virtual std::vector<std::vector<rtr::Color> >& rtr::IRenderer::getPixels ( ) [pure virtual]
```

Gets the pixels of the renderer based on the configuration file.

##### Returns

Each pixels of the image.

Implemented in [rtr::ARenderer](#).

#### 4.16.2.4 `getResolution()`

```
virtual Resolution& rtr::IRenderer::getResolution ( ) [pure virtual]
```

Gets the resolution of the renderer based on the configuration file.

##### Returns

The resolution of the renderer using the [Resolution](#) class.

Implemented in [rtr::ARenderer](#).

#### 4.16.2.5 `getType()`

```
virtual const RendererType& rtr::IRenderer::getType ( ) const [pure virtual]
```

Gets the type of the renderer based on the configuration file.

##### Returns

The type of the renderer, using the enum class `RendererType`.

Implemented in [rtr::ARenderer](#).

#### 4.16.2.6 `render()`

```
virtual void rtr::IRenderer::render (
    const std::vector< std::unique_ptr< AShape >> & shapes,
    const std::vector< std::unique_ptr< ALight >> & lights,
    const Camera & camera ) [pure virtual]
```

Renders the scene based on the shapes, lights and camera.

##### Parameters

<i>shapes</i>	The shapes of the scene.
<i>lights</i>	The lights of the scene.
<i>camera</i>	The camera of the scene.

Implemented in [rtr::SFML](#), and [rtr::PPM](#).

#### 4.16.2.7 `setName()`

```
virtual void rtr::IRenderer::setName (
```

```
const std::string & name ) [pure virtual]
```

Sets the name of the renderer.

#### Parameters

<i>name</i>	The name of the renderer, based on the configuration file.
-------------	--

Implemented in [rtr::ARenderer](#).

#### 4.16.2.8 setPixels()

```
virtual void rtr::IRenderer::setPixels (
    const std::vector< std::vector< rtr::Color >> & pixels ) [pure virtual]
```

Sets the pixels of the renderer.

#### Parameters

<i>pixels</i>	The pixels of the renderer.
---------------	-----------------------------

Implemented in [rtr::ARenderer](#).

#### 4.16.2.9 setType()

```
virtual void rtr::IRenderer::setType (
    const RendererType & rendererType ) [pure virtual]
```

Sets the type of the renderer.

#### Parameters

<i>rendererType</i>	The type of the renderer (defined in the RendererType enum class).
---------------------	--

Implemented in [rtr::ARenderer](#).

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

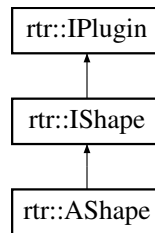
- App/include/RayTracer/Abstraction/[IRenderer.hpp](#)

## 4.17 rtr::IShape Class Reference

An interface used to get the shape's parameters based on the configuration file.

```
#include <IShape.hpp>
```

Inheritance diagram for rtr::IShape:



## Public Member Functions

- virtual void [setType](#) (const ShapeType &type)=0  
*Sets the type of the shape (sphere, plane, cone...).*
- virtual void [setMaterial](#) (std::unique\_ptr< [AMaterial](#) > material)=0  
*Sets the material of the shape.*
- virtual void [setRadius](#) (const double &radius)=0  
*Sets the radius of the shape.*
- virtual void [setHeight](#) (const double &height)=0  
*Sets the height of the shape.*
- virtual const ShapeType & [getType](#) () const =0  
*Gets the type of the shape.*
- virtual [AMaterial](#) & [getMaterial](#) ()=0  
*Gets the material of the shape.*
- virtual [Vector](#) & [getPosition](#) ()=0  
*Gets the position of the shape.*
- virtual [Vector](#) & [getNormal](#) ()=0  
*Gets the normal of the shape.*
- virtual [Vector](#) & [getRotation](#) ()=0  
*Gets the rotation of the shape, used to create the shape.*
- virtual const double & [getRadius](#) () const =0  
*Gets the radius of the shape.*
- virtual const double & [getHeight](#) () const =0  
*Gets the height of the shape.*
- virtual bool [hits](#) (std::pair< [Vector](#), [Vector](#) > ray, [RayHit](#) &hit)=0  
*Checks if the ray hits the shape, used to render the output file.*
- virtual [Vector](#) [getDistance](#) (const [Vector](#) &point)=0  
*Gets the distance between the shape and a point.*

### 4.17.1 Detailed Description

An interface used to get the shape's parameters based on the configuration file.

### 4.17.2 Member Function Documentation



#### 4.17.2.1 getDistance()

```
virtual Vector rtr::IShape::getDistance (
    const Vector & point ) [pure virtual]
```

Gets the distance between the shape and a point.

##### Parameters

<i>point</i>	The point to check the distance with.
--------------	---------------------------------------

##### Returns

The distance between the shape and the point.

Implemented in [rtr::AShape](#).

#### 4.17.2.2 getHeight()

```
virtual const double& rtr::IShape::getHeight ( ) const [pure virtual]
```

Gets the height of the shape.

##### Returns

The height of the shape as a double.

Implemented in [rtr::AShape](#).

#### 4.17.2.3 getMaterial()

```
virtual AMaterial& rtr::IShape::getMaterial ( ) [pure virtual]
```

Gets the material of the shape.

##### Returns

The material of the shape, using the [AMaterial](#) class.

Implemented in [rtr::AShape](#).

#### 4.17.2.4 getNormal()

```
virtual Vector& rtr::IShape::getNormal ( ) [pure virtual]
```

Gets the normal of the shape.

##### Returns

The normal of the shape, using the [Vector](#) class.

Implemented in [rtr::AShape](#).

#### 4.17.2.5 getPosition()

```
virtual Vector& rtr::IShape::getPosition ( ) [pure virtual]
```

Gets the position of the shape.

##### Returns

The position of the shape, using the [Vector](#) class.

Implemented in [rtr::AShape](#).

#### 4.17.2.6 getRadius()

```
virtual const double& rtr::IShape::getRadius ( ) const [pure virtual]
```

Gets the radius of the shape.

##### Returns

The radius of the shape as a double, used to check the size of the shape.

Implemented in [rtr::AShape](#).

#### 4.17.2.7 getRotation()

```
virtual Vector& rtr::IShape::getRotation ( ) [pure virtual]
```

Gets the rotation of the shape, used to create the shape.

##### Returns

The rotation of the shape, using the [Vector](#) class.

Implemented in [rtr::AShape](#).

#### 4.17.2.8 getType()

```
virtual const ShapeType& rtr::IShape::getType ( ) const [pure virtual]
```

Gets the type of the shape.

##### Returns

The type of the shape, using the ShapeType enum class.

Implemented in [rtr::AShape](#).

#### 4.17.2.9 hits()

```
virtual bool rtr::IShape::hits (
    std::pair< Vector, Vector > ray,
    RayHit & hit ) [pure virtual]
```

Checks if the ray hits the shape, used to render the output file.

##### Parameters

<i>ray</i>	A pair of vectors, to get the rays of the "camera".
<i>hit</i>	The hit of the ray, used to check if the ray hits the shape.

##### Returns

A boolean, true if the ray hits the shape, false otherwise.

#### 4.17.2.10 setHeight()

```
virtual void rtr::IShape::setHeight (
    const double & height ) [pure virtual]
```

Sets the height of the shape.

##### Parameters

<i>height</i>	The height of the shape.
---------------	--------------------------

Implemented in [rtr::AShape](#).

#### 4.17.2.11 setMaterial()

```
virtual void rtr::IShape::setMaterial (
    std::unique_ptr< AMaterial > material ) [pure virtual]
```

Sets the material of the shape.

##### Parameters

<i>material</i>	The material of the shape (transparency, reflectivity...).
-----------------	--

Implemented in [rtr::AShape](#).

#### 4.17.2.12 setRadius()

```
virtual void rtr::IShape::setRadius (
    const double & radius ) [pure virtual]
```

Sets the radius of the shape.

##### Parameters

<i>radius</i>	The radius of the shape.
---------------	--------------------------

Implemented in [rtr::AShape](#).

#### 4.17.2.13 setType()

```
virtual void rtr::IShape::setType (
    const ShapeType & type ) [pure virtual]
```

Sets the type of the shape (sphere, plane, cone...).

##### Parameters

<i>type</i>	The type of the shape.
-------------	------------------------

Implemented in [rtr::AShape](#).

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

- [App/include/RayTracer/Abstraction/IShape.hpp](#)

## 4.18 rtr::LightFactory Class Reference

A factory class for the lights.

```
#include <Light.hpp>
```

### Static Public Member Functions

- static std::unique\_ptr< [ALight](#) > [createLight](#) (const [Color](#) &color, const float &intensity)  
*Creates a light based on the color and intensity (specific to the ambient light).*
- static std::unique\_ptr< [ALight](#) > [createLight](#) (const LightType &type, const [Color](#) &color, const float &intensity, const [Vector](#) &vector)  
*Creates a light based on the type, color, intensity and a vector (used for directional and point lights).*

### 4.18.1 Detailed Description

A factory class for the lights.

### 4.18.2 Member Function Documentation

#### 4.18.2.1 createLight() [1/2]

```
static std::unique_ptr<ALight> rtr::LightFactory::createLight (
    const Color & color,
    const float & intensity ) [static]
```

Creates a light based on the color and intensity (specific to the ambient light).

#### Parameters

<i>color</i>	The color of the light.
<i>intensity</i>	The intensity of the light.

#### Returns

A unique pointer to the light.

#### 4.18.2.2 createLight() [2/2]

```
static std::unique_ptr<ALight> rtr::LightFactory::createLight (
    const LightType & type,
```

```
const Color & color,
const float & intensity,
const Vector & vector ) [static]
```

Creates a light based on the type, color, intensity and a vector (used for directional and point lights).

#### Parameters

<i>type</i>	The type of the light (defined in the LightType enum class).
<i>color</i>	The color of the light.
<i>intensity</i>	The intensity of the light.
<i>vector</i>	A vector to get the position of the light.

#### Returns

A unique pointer to the light.

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

- App/include/RayTracer/Factory/Light.hpp

## 4.19 rtr::MaterialFactory Class Reference

A factory class for the materials of the shapes.

```
#include <Material.hpp>
```

### Static Public Member Functions

- static std::unique\_ptr<AMaterial> createMaterial (const MaterialType &type, const float &floatValue)  
*Creates a material based on the type and the color.*

#### 4.19.1 Detailed Description

A factory class for the materials of the shapes.

#### 4.19.2 Member Function Documentation

##### 4.19.2.1 createMaterial()

```
static std::unique_ptr<AMaterial> rtr::MaterialFactory::createMaterial (
    const MaterialType & type,
    const float & floatValue ) [static]
```

Creates a material based on the type and the color.

## Parameters

<i>type</i>	The type of the material (defined in the MaterialType enum class).
<i>floatValue</i>	The value of the transparency and reflectiveness to set.

## Returns

A unique pointer to the material.

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

- App/include/RayTracer/Factory/[Material.hpp](#)

## 4.20 rtr::Parser Class Reference

Class dedicated to the parsing of configuration files and command-line arguments.

```
#include <Parser.hpp>
```

### Classes

- class [ParserException](#)  
*Exception class for errors in the parsers.*

### Static Public Member Functions

- static int [parseArgs](#) (const std::string &filePath)  
*Parses command-line arguments.*
- static std::unique\_ptr< [rtr::Scene](#) > [parseFile](#) (const std::string &filePath)  
*Parses a configuration file and returns a [Scene](#) object.*
- static void [parseRenderer](#) (const libconfig::Setting &renderer, [Scene](#) &scene)  
*Parses the renderer settings from a configuration file.*
- static void [parseCamera](#) (const libconfig::Setting &camera, [Scene](#) &scene)  
*Parses the camera settings from a configuration file.*
- static ShapeType [parseShapeType](#) (const std::string &type)  
*Parses the shape type from a string.*
- static void [parseShapes](#) (const libconfig::Setting &shapesSetting, [Scene](#) &scene)  
*Parses the shapes settings from a configuration file.*
- static std::unique\_ptr< [AMaterial](#) > [parseMaterial](#) (const libconfig::Setting &materialSetting)  
*Parses the material settings from a configuration file.*
- static LightType [parseLightType](#) (const std::string &type)  
*Parses the light settings from a configuration file.*
- static void [parseLights](#) (const libconfig::Setting &lightsSetting, [Scene](#) &scene)  
*Parses the light settings from a configuration file.*
- template<typename T, typename ConversionFunc >  
static T [getVector](#) (const libconfig::Setting &setting, ConversionFunc convert)  
*Templated function to get a vector from a configuration file setting.*
- template<typename T >  
static T [convertInt](#) (const libconfig::Setting &setting)  
*Templated function to convert an integer value from a configuration file setting.*

### 4.20.1 Detailed Description

Class dedicated to the parsing of configuration files and command-line arguments.

### 4.20.2 Member Function Documentation

#### 4.20.2.1 convertInt()

```
template<typename T >
static T rtr::Parser::convertInt (
    const libconfig::Setting & setting ) [static]
```

Templated function to convert an integer value from a configuration file setting.

##### Template Parameters

<i>T</i>	The type of the value.
----------	------------------------

##### Parameters

<i>setting</i>	The setting in the configuration file.
----------------	--

##### Returns

The parsed value.

#### 4.20.2.2 getVector()

```
template<typename T , typename ConversionFunc >
static T rtr::Parser::getVector (
    const libconfig::Setting & setting,
    ConversionFunc convert ) [static]
```

Templated function to get a vector from a configuration file setting.

##### Template Parameters

<i>T</i>	The type of the vector.
<i>ConversionFunc</i>	The function to convert the vector elements.

##### Parameters

<i>setting</i>	The setting in the configuration file.
----------------	--



## Parameters

<i>convert</i>	The conversion function.
----------------	--------------------------

## Returns

The parsed vector.

### 4.20.2.3 parseArgs()

```
static int rtr::Parser::parseArgs (
    const std::string & filePath ) [static]
```

Parses command-line arguments.

## Parameters

<i>filePath</i>	The path to the configuration file.
-----------------	-------------------------------------

## Returns

0 on success, 1 on failure.

### 4.20.2.4 parseCamera()

```
static void rtr::Parser::parseCamera (
    const libconfig::Setting & camera,
    Scene & scene ) [static]
```

Parses the camera settings from a configuration file.

## Parameters

<i>camera</i>	The camera settings in the configuration file.
<i>scene</i>	The <a href="#">Scene</a> object to update with the parsed settings.

### 4.20.2.5 parseFile()

```
static std::unique_ptr<rtr::Scene> rtr::Parser::parseFile (
    const std::string & filePath ) [static]
```

Parses a configuration file and returns a [Scene](#) object.

**Parameters**

<i>filePath</i>	The path to the configuration file.
-----------------	-------------------------------------

**Returns**

A `unique_ptr` to a [Scene](#) object.

**4.20.2.6 parseLights()**

```
static void rtr::Parser::parseLights (
    const libconfig::Setting & lightsSetting,
    Scene & scene ) [static]
```

Parses the light settings from a configuration file.

**Parameters**

<i>lightsSetting</i>	The lights settings in the configuration files.
<i>scene</i>	The <a href="#">Scene</a> object to update with the parsed settings.

**4.20.2.7 parseLightType()**

```
static LightType rtr::Parser::parseLightType (
    const std::string & type ) [static]
```

Parses the light settings from a configuration file.

**Parameters**

<i>type</i>	The light type as a string.
-------------	-----------------------------

**Returns**

The parsed light type.

**4.20.2.8 parseMaterial()**

```
static std::unique_ptr<AMaterial> rtr::Parser::parseMaterial (
    const libconfig::Setting & materialSetting ) [static]
```

Parses the material settings from a configuration file.

## Parameters

<i>materialSetting</i>	The material settings in the configuration file.
------------------------	--

## Returns

A `unique_ptr` to a Material object.

#### 4.20.2.9 parseRenderer()

```
static void rtr::Parser::parseRenderer (
    const libconfig::Setting & renderer,
    Scene & scene ) [static]
```

Parses the renderer settings from a configuration file.

## Parameters

<i>renderer</i>	The renderer settings in the configuration file.
<i>scene</i>	The <a href="#">Scene</a> object to update with the parsed settings.

#### 4.20.2.10 parseShapes()

```
static void rtr::Parser::parseShapes (
    const libconfig::Setting & shapesSetting,
    Scene & scene ) [static]
```

Parses the shapes settings from a configuration file.

## Parameters

<i>shapesSetting</i>	The shapes settings in the configuration file.
<i>scene</i>	The <a href="#">Scene</a> object to update with the parsed settings.

#### 4.20.2.11 parseShapeType()

```
static ShapeType rtr::Parser::parseShapeType (
    const std::string & type ) [static]
```

Parses the shape type from a string.

## Parameters

<i>type</i>	The shape type as a string.
-------------	-----------------------------

## Returns

The parsed shape type.

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

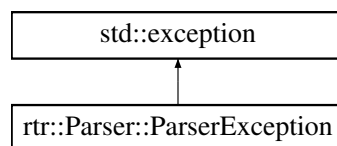
- [App/include/RayTracer/Parser.hpp](#)

## 4.21 rtr::Parser::ParserException Class Reference

Exception class for errors in the parsers.

```
#include <Parser.hpp>
```

Inheritance diagram for rtr::Parser::ParserException:



### Public Member Functions

- **ParserException** (std::string msg)
- **ParserException** (const [ParserException](#) &)=delete
- [ParserException](#) & **operator=** (const [ParserException](#) &)=delete
- **ParserException** (const [ParserException](#) &&)=delete
- [ParserException](#) & **operator=** (const [ParserException](#) &&)=delete
- const char \* [what](#) () const noexcept override

*Returns the error message.*

#### 4.21.1 Detailed Description

Exception class for errors in the parsers.

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

- [App/include/RayTracer/Parser.hpp](#)

## 4.22 rtr::PluginLoader Class Reference

A class to load the plugins.

```
#include <Plugin.hpp>
```

### Public Types

- using **PluginCreator** = std::unique\_ptr< [IPlugin](#) >(\*)()

### Public Member Functions

- template<typename T >  
std::unique\_ptr< T > [getPlugin](#) (const std::string &pluginName)  
*Gets the plugin based on the name.*
- void **closePlugins** ()

### Static Public Member Functions

- static [PluginLoader](#) & [getInstance](#) ()  
*Gets the instance of the plugin loader.*

#### 4.22.1 Detailed Description

A class to load the plugins.

#### 4.22.2 Member Function Documentation

##### 4.22.2.1 getInstance()

```
static PluginLoader& rtr::PluginLoader::getInstance ( ) [inline], [static]
```

Gets the instance of the plugin loader.

##### Returns

A reference to the plugin loader.

##### 4.22.2.2 getPlugin()

```
template<typename T >  
std::unique_ptr<T> rtr::PluginLoader::getPlugin (  
    const std::string & pluginName )
```

Gets the plugin based on the name.

## Parameters

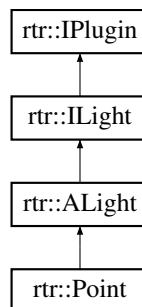
<code>pluginName</code>	The name of the plugin.
-------------------------	-------------------------

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

- App/include/RayTracer/Loader/[Plugin.hpp](#)

## 4.23 rtr::Point Class Reference

Inheritance diagram for rtr::Point:



### Public Member Functions

- `std::string getPluginName ()` const override  
*Gets the name of the plugin.*
- `Color LightColor (const Vector &normal, const Color &col)` override  
*Creates light effects based on the light type.*
- `Vector & getDirection ()` override  
*Gets the direction of the light based on the configuration file.*

### 4.23.1 Member Function Documentation

#### 4.23.1.1 `getPluginName()`

```
std::string rtr::Point::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

#### Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

#### 4.23.1.2 LightColor()

```
Color rtr::Point::LightColor (
    const Vector & normal,
    const Color & col ) [override], [virtual]
```

Creates light effects based on the light type.

## Parameters

<i>normal</i>	The normal of the shape.
<i>col</i>	The current color of the shape.

## Returns

The new color of the shape with the light effects.

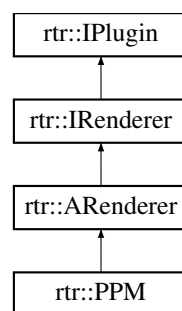
Implements [rtr::ILight](#).

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

- App/plugins/Light/Point/include/RayTracer/Point.hpp

## 4.24 rtr::PPM Class Reference

Inheritance diagram for rtr::PPM:



### Public Member Functions

- `std::string getPluginName ()` const override  
*Gets the name of the plugin.*
- `void render (const std::vector< std::unique_ptr< AShape >> &shapes, const std::vector< std::unique_ptr< ALight >> &lights, const Camera &camera)` override  
*Renders the scene based on the shapes, lights and camera.*
- `void writePixels (const Color color, const std::size_t width, const std::size_t height)`
- `void writeToFile (const std::string &width, const std::string &height)`
- `bool isShadowed (const Vector &lightDir, const Vector &point, const std::vector< std::unique_ptr< AShape >> &shapes)`

### Static Public Member Functions

- `static std::string getHeader (const std::string &width, const std::string &height)`



## 4.24.1 Member Function Documentation

### 4.24.1.1 getPluginName()

```
std::string rtr::PPM::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

#### Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

### 4.24.1.2 render()

```
void rtr::PPM::render (
    const std::vector< std::unique_ptr< AShape >> & shapes,
    const std::vector< std::unique_ptr< ALight >> & lights,
    const Camera & camera ) [override], [virtual]
```

Renders the scene based on the shapes, lights and camera.

#### Parameters

<i>shapes</i>	The shapes of the scene.
<i>lights</i>	The lights of the scene.
<i>camera</i>	The camera of the scene.

Implements [rtr::IRenderer](#).

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

- App/plugins/Renderer/PPM/include/RayTracer/PPM.hpp

## 4.25 ray\_hit\_s Struct Reference

A struct representing a ray hit in 3D space.

```
#include <RayHit.hpp>
```

### 4.25.1 Detailed Description

A struct representing a ray hit in 3D space.

Type alias for a ray hit.

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

- App/include/RayTracer/Utils/[RayHit.hpp](#)

## 4.26 rtr::RayHit Class Reference

A class representing a ray hit in 3D space.

```
#include <RayHit.hpp>
```

### Public Member Functions

- `const ray_hit_t & getRayHit ()` const noexcept
- `void setRayHit (const ray_hit_t &ray_hit)` noexcept  
*Sets the ray hit data.*
- `void setRayHit (const Vector &point, const Vector &normal, const double &distance)` noexcept  
*Sets the ray hit data.*
- `void setPoint (const Vector &point)` noexcept  
*Sets the point of intersection.*
- `void setNormal (const Vector &normal)` noexcept  
*Sets the normal vector at the point of intersection.*
- `void setDistance (const double &distance)` noexcept  
*Sets the distance from the ray origin to the point of intersection.*

### 4.26.1 Detailed Description

A class representing a ray hit in 3D space.

### 4.26.2 Member Function Documentation

#### 4.26.2.1 setDistance()

```
void rtr::RayHit::setDistance (  
    const double & distance ) [inline], [noexcept]
```

Sets the distance from the ray origin to the point of intersection.

## Parameters

<i>distance</i>	The distance to set.
-----------------	----------------------

### 4.26.2.2 setNormal()

```
void rtr::RayHit::setNormal (
    const Vector & normal ) [inline], [noexcept]
```

Sets the normal vector at the point of intersection.

## Parameters

<i>normal</i>	The normal vector to set.
---------------	---------------------------

### 4.26.2.3 setPoint()

```
void rtr::RayHit::setPoint (
    const Vector & point ) [inline], [noexcept]
```

Sets the point of intersection.

## Parameters

<i>point</i>	The point of intersection to set.
--------------	-----------------------------------

### 4.26.2.4 setRayHit() [1/2]

```
void rtr::RayHit::setRayHit (
    const ray_hit_t & ray_hit ) [inline], [noexcept]
```

Sets the ray hit data.

## Parameters

<i>ray_hit</i>	The ray hit data to set
----------------	-------------------------

#### 4.26.2.5 setRayHit() [2/2]

```
void rtr::RayHit::setRayHit (
    const Vector & point,
    const Vector & normal,
    const double & distance ) [inline], [noexcept]
```

Sets the ray hit data.

##### Parameters

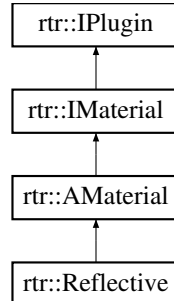
<i>point</i>	The point of intersection.
<i>normal</i>	The normal vector at the point of intersection.
<i>distance</i>	The distance from the ray origin to the point of intersection.

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

- App/include/RayTracer/Utils/[RayHit.hpp](#)

## 4.27 rtr::Reflective Class Reference

Inheritance diagram for rtr::Reflective:



### Public Member Functions

- void [applyMaterial](#) (Color \*color) override  
*Applies the material to the shape (transparency and reflectiveness).*
- std::string [getPluginName](#) () const override  
*Gets the name of the plugin.*

#### 4.27.1 Member Function Documentation

#### 4.27.1.1 getPluginName()

```
std::string rtr::Reflective::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

##### Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

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

- App/plugins/Material/Reflective/include/RayTracer/Reflective.hpp

## 4.28 rtr::RendererFactory Class Reference

A factory class to create the renderers.

```
#include <Renderer.hpp>
```

### Static Public Member Functions

- static std::unique\_ptr<[ARenderer](#)> [createRenderer](#) (const RendererType &type, const std::string &name, const [Resolution](#) &resolution, const [Color](#) &backgroundColor)

*Creates a renderer based on the type, name, resolution and background color.*

#### 4.28.1 Detailed Description

A factory class to create the renderers.

#### 4.28.2 Member Function Documentation

##### 4.28.2.1 createRenderer()

```
static std::unique_ptr<ARenderer> rtr::RendererFactory::createRenderer (
    const RendererType & type,
    const std::string & name,
    const Resolution & resolution,
    const Color & backgroundColor ) [static]
```

Creates a renderer based on the type, name, resolution and background color.

**Parameters**

<i>type</i>	The type of the renderer (defined in the <code>RendererType</code> enum class).
<i>name</i>	A string of the renderer's name.
<i>resolution</i>	The resolution of the renderer.
<i>backgroundColor</i>	The background color of the renderer.

**Returns**

A unique pointer to the renderer.

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

- `App/include/RayTracer/Factory/Renderer.hpp`

## 4.29 rtr::Resolution Class Reference

Class representing the resolution of an image.

```
#include <Resolution.hpp>
```

**Public Member Functions**

- **Resolution** (const uint16\_t &width, const uint16\_t &height)
- **Resolution** (const resolution\_t &resolution)
- void **setWidth** (const uint16\_t &width)  
*Sets the width of the resolution.*
- void **setHeight** (const uint16\_t &height)  
*Sets the height of the resolution.*
- void **setResolution** (const uint16\_t &width, const uint16\_t &height)  
*Sets the resolution to the given width and height.*
- void **setResolution** (const resolution\_t &resolution)  
*Sets the resolution to the given resolution struct.*
- uint16\_t **getWidth** () const  
*Gets the width of the resolution.*
- uint16\_t **getHeight** () const  
*Gets the height of the resolution.*
- resolution\_t **getValue** () const  
*Gets the resolution as a struct.*

### 4.29.1 Detailed Description

Class representing the resolution of an image.

## 4.29.2 Member Function Documentation

### 4.29.2.1 getHeight()

```
uint16_t rtr::Resolution::getHeight ( ) const [inline]
```

Gets the height of the resolution.

#### Returns

The height of the resolution.

### 4.29.2.2 getValue()

```
resolution_t rtr::Resolution::getValue ( ) const [inline]
```

Gets the resolution as a struct.

#### Returns

The resolution struct.

### 4.29.2.3 getWidth()

```
uint16_t rtr::Resolution::getWidth ( ) const [inline]
```

Gets the width of the resolution.

#### Returns

The width of the resolution.

### 4.29.2.4 setHeight()

```
void rtr::Resolution::setHeight (
    const uint16_t & height ) [inline]
```

Sets the height of the resolution.

## Parameters

<i>height</i>	The height to set.
---------------	--------------------

**4.29.2.5 setResolution() [1/2]**

```
void rtr::Resolution::setResolution (
    const resolution_t & resolution ) [inline]
```

Sets the resolution to the given resolution struct.

## Parameters

<i>resolution</i>	The resolution struct to set.
-------------------	-------------------------------

**4.29.2.6 setResolution() [2/2]**

```
void rtr::Resolution::setResolution (
    const uint16_t & width,
    const uint16_t & height ) [inline]
```

Sets the resolution to the given width and height.

## Parameters

<i>width</i>	The width of the resolution.
<i>height</i>	The height of the resolution.

**4.29.2.7 setWidth()**

```
void rtr::Resolution::setWidth (
    const uint16_t & width ) [inline]
```

Sets the width of the resolution.

## Parameters

<i>width</i>	The width to set.
--------------	-------------------

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

- App/include/RayTracer/Utils/[Resolution.hpp](#)



## 4.30 resolution\_s Struct Reference

A struct representing the resolution of an image.

```
#include <Resolution.hpp>
```

### 4.30.1 Detailed Description

A struct representing the resolution of an image.

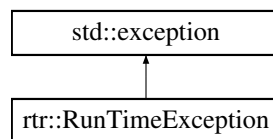
Type alias for the resolution of an image.

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

- App/include/RayTracer/Utils/[Resolution.hpp](#)

## 4.31 rtr::RunTimeException Class Reference

Inheritance diagram for rtr::RunTimeException:



### Public Member Functions

- **RunTimeException** (std::string msg)
- **RunTimeException** (const [RunTimeException](#) &)=delete
- [RunTimeException](#) & **operator=** (const [RunTimeException](#) &)=delete
- **RunTimeException** (const [RunTimeException](#) &&)=delete
- [RunTimeException](#) & **operator=** (const [RunTimeException](#) &&)=delete
- const char \* **what** () const noexcept override

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

- App/include/RayTracer/Exception/RunTime.hpp

## 4.32 rtr::Scene Class Reference

A class to represent the scene.

```
#include <Scene.hpp>
```

## Public Member Functions

- void [setCamera](#) (const [Camera](#) &camera)  
*Sets the camera of the scene.*
- void [setRenderer](#) (std::unique\_ptr< [ARenderer](#) > renderer)  
*Sets the renderer of the scene.*
- void [addShape](#) (std::unique\_ptr< [AShape](#) > shape)  
*Adds a shape to the scene.*
- void [addLight](#) (std::unique\_ptr< [ALight](#) > light)  
*Adds a light to the scene.*
- [Camera](#) & [getCamera](#) ()  
*Gets the camera of the scene.*
- const std::unique\_ptr< [ARenderer](#) > & [getRenderer](#) () const  
*Gets the renderer of the scene.*
- const std::vector< std::unique\_ptr< [AShape](#) > > & [getShapes](#) () const  
*Gets the shapes of the scene.*
- const std::vector< std::unique\_ptr< [ALight](#) > > & [getLights](#) () const  
*Gets the lights of the scene.*

### 4.32.1 Detailed Description

A class to represent the scene.

### 4.32.2 Member Function Documentation

#### 4.32.2.1 addLight()

```
void rtr::Scene::addLight (
    std::unique_ptr< ALight > light ) [inline]
```

Adds a light to the scene.

##### Parameters

<i>light</i>	The light to add, which can be a point light, a directional light and an ambient light.
--------------	---

#### 4.32.2.2 addShape()

```
void rtr::Scene::addShape (
    std::unique_ptr< AShape > shape ) [inline]
```

Adds a shape to the scene.

## Parameters

<i>shape</i>	The shape to add, which can be a sphere, a plane and a cone.
--------------	--

**4.32.2.3 getLights()**

```
const std::vector<std::unique_ptr<ALight> >& rtr::Scene::getLights ( ) const [inline]
```

Gets the lights of the scene.

## Returns

A vector of unique pointers to the lights.

**4.32.2.4 getRenderer()**

```
const std::unique_ptr<ARenderer>& rtr::Scene::getRenderer ( ) const [inline]
```

Gets the renderer of the scene.

## Returns

A unique pointer to the renderer.

**4.32.2.5 getShapes()**

```
const std::vector<std::unique_ptr<AShape> >& rtr::Scene::getShapes ( ) const [inline]
```

Gets the shapes of the scene.

## Returns

A vector of unique pointers to the shapes.

**4.32.2.6 setCamera()**

```
void rtr::Scene::setCamera (
    const Camera & camera ) [inline]
```

Sets the camera of the scene.

## Parameters

<i>camera</i>	The camera to set.
---------------	--------------------

**4.32.2.7 setRenderer()**

```
void rtr::Scene::setRenderer (
    std::unique_ptr< ARenderer > renderer ) [inline]
```

Sets the renderer of the scene.

## Parameters

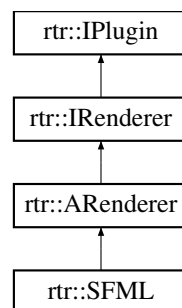
<i>renderer</i>	The renderer to set.
-----------------	----------------------

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

- App/include/RayTracer/Scene/[Scene.hpp](#)

**4.33 rtr::SFML Class Reference**

Inheritance diagram for rtr::SFML:

**Public Member Functions**

- `std::string getPluginName ()` const override  
*Gets the name of the plugin.*
- `void render (const std::vector< std::unique_ptr< AShape >> &shapes, const std::vector< std::unique_ptr< ALight >> &lights, const Camera &camera)` override  
*Renders the scene based on the shapes, lights and camera.*

**4.33.1 Member Function Documentation**

4.33.1.1 `getPluginName()`

```
std::string rtr::SFML::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

## Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

4.33.1.2 `render()`

```
void rtr::SFML::render (
    const std::vector< std::unique_ptr< AShape >> & shapes,
    const std::vector< std::unique_ptr< ALight >> & lights,
    const Camera & camera ) [override], [virtual]
```

Renders the scene based on the shapes, lights and camera.

## Parameters

<i>shapes</i>	The shapes of the scene.
<i>lights</i>	The lights of the scene.
<i>camera</i>	The camera of the scene.

Implements [rtr::IRenderer](#).

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

- App/plugins/Renderer/SFML/include/RayTracer/SFML.hpp

4.34 `rtr::ShapeFactory` Class Reference

A factory class for the shapes.

```
#include <Shape.hpp>
```

## Static Public Member Functions

- static `std::unique_ptr< AShape > createShape` (const `Vector` &position, const `Vector` &normal)  
*Creates a shape based on the position and normal (for the plane).*
- static `std::unique_ptr< AShape > createShape` (const `Vector` &position, const double &radius)  
*Creates a shape based on the position and radius (for the sphere).*
- static `std::unique_ptr< AShape > createShape` (const ShapeType &type, const `Vector` &position, const `Vector` &rotation, const double &radius, const double &height)  
*Creates a shape based on the type position, rotation, radius and height.*

### 4.34.1 Detailed Description

A factory class for the shapes.

### 4.34.2 Member Function Documentation

#### 4.34.2.1 createShape() [1/3]

```
static std::unique_ptr<AShape> rtr::ShapeFactory::createShape (
    const ShapeType & type,
    const Vector & position,
    const Vector & rotation,
    const double & radius,
    const double & height ) [static]
```

Creates a shape based on the type position, rotation, radius and height.

##### Parameters

<i>type</i>	The type of the shape (defined in the ShapeType enum class).
<i>position</i>	Used to get the position of the shape.
<i>rotation</i>	Used to get the rotation of the shape.
<i>radius</i>	Used to get the radius of the shape.
<i>height</i>	Used to get the height of the shape.

##### Returns

A unique pointer to the shape.

#### 4.34.2.2 createShape() [2/3]

```
static std::unique_ptr<AShape> rtr::ShapeFactory::createShape (
    const Vector & position,
    const double & radius ) [static]
```

Creates a shape based on the position and radius (for the sphere).

##### Parameters

<i>position</i>	Used to get the position of the shape.
<i>radius</i>	Used to get the radius of the shape.

## Returns

A unique pointer to the shape.

## 4.34.2.3 createShape() [3/3]

```
static std::unique_ptr<AShape> rtr::ShapeFactory::createShape (
    const Vector & position,
    const Vector & normal ) [static]
```

Creates a shape based on the position and normal (for the plane).

## Parameters

<i>position</i>	Used to get the position of the shape.
<i>normal</i>	Used to get the normal of the shape.

## Returns

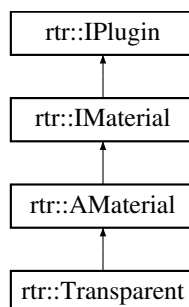
A unique pointer to the shape.

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

- [App/include/RayTracer/Factory/Shape.hpp](#)

## 4.35 rtr::Transparent Class Reference

Inheritance diagram for rtr::Transparent:



## Public Member Functions

- void [applyMaterial](#) (Color \*color) override  
*Applies the material to the shape (transparency and reflectiveness).*
- std::string [getPluginName](#) () const override  
*Gets the name of the plugin.*

### 4.35.1 Member Function Documentation

#### 4.35.1.1 getPluginName()

```
std::string rtr::Transparent::getPluginName ( ) const [inline], [override], [virtual]
```

Gets the name of the plugin.

##### Returns

A string of the plugin's name, defined as const expressions.

Implements [rtr::IPlugin](#).

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

- App/plugins/Material/Transparent/include/RayTracer/Transparent.hpp

## 4.36 rtr::Vector Class Reference

### Public Member Functions

- **Vector** (const double x, const double y, const double z)
- **Vector** (const vector\_t position)
- void [setX](#) (const double x)  
*Sets the x-component of the vector.*
- void [setY](#) (const double y)  
*Sets the y-component of the vector.*
- void [setZ](#) (const double z)  
*Sets the z-component of the vector.*
- void [setVector](#) (const double x, const double y, const double z)  
*Sets the vector to the given x, y, and z values.*
- void [setVector](#) (const vector\_t &position)  
*Sets the vector to the given vector struct.*
- double [getX](#) () const  
*Gets the x-component of the vector.*
- double [getY](#) () const  
*Gets the y-component of the vector.*
- double [getZ](#) () const  
*Gets the z-component of the vector.*
- vector\_t [getValue](#) () const  
*Gets the vector as a struct.*
- **Vector operator+** (const **Vector** &other) const  
*Adds two vectors.*
- **Vector operator+** (const double scalar) const  
*Adds a scalar to the vector.*



- **Vector operator-** (const **Vector** &other) const  
*Subtracts two vectors.*
- **Vector operator\*** (const **Vector** &other) const  
*Multiplies two vectors.*
- **Vector operator\*** (const double scalar) const  
*Multiplies the vector by a scalar.*
- **Vector operator/** (const double scalar) const  
*Divides the vector by a scalar.*
- double **length** () const  
*Gets the length of the vector.*
- double **dot** (const **Vector** &other) const  
*Calculates the dot product of the vector and another vector.*
- **Vector cross** (const **Vector** &other) const  
*Calculates the cross product of the vector and another vector.*
- **Vector normalize** () const  
*Normalizes the vector.*

### 4.36.1 Member Function Documentation

#### 4.36.1.1 cross()

```
Vector rtr::Vector::cross (
    const Vector & other ) const [inline]
```

Calculates the cross product of the vector and another vector.

##### Parameters

<i>other</i>	The other vector to calculate the cross product with.
--------------	---

##### Returns

The cross product of the two vectors.

#### 4.36.1.2 dot()

```
double rtr::Vector::dot (
    const Vector & other ) const [inline]
```

Calculates the dot product of the vector and another vector.

##### Parameters

<i>other</i>	The other vector to calculate the dot product with.
--------------	---

**Returns**

The dot product of the two vectors.

**4.36.1.3   getValue()**

```
vector_t rtr::Vector::getValue ( ) const [inline]
```

Gets the vector as a struct.

**Returns**

The vector struct.

**4.36.1.4   getX()**

```
double rtr::Vector::getX ( ) const [inline]
```

Gets the x-component of the vector.

**Returns**

The x-component of the vector.

**4.36.1.5   getY()**

```
double rtr::Vector::getY ( ) const [inline]
```

Gets the y-component of the vector.

**Returns**

The y-component of the vector.

**4.36.1.6   getZ()**

```
double rtr::Vector::getZ ( ) const [inline]
```

Gets the z-component of the vector.

**Returns**

The z-component of the vector.

#### 4.36.1.7 length()

```
double rtr::Vector::length ( ) const [inline]
```

Gets the length of the vector.

##### Returns

The length of the vector.

#### 4.36.1.8 normalize()

```
Vector rtr::Vector::normalize ( ) const [inline]
```

Normalizes the vector.

##### Returns

The normalized vector.

#### 4.36.1.9 operator\*() [1/2]

```
Vector rtr::Vector::operator* (
    const double scalar ) const [inline]
```

Multiplies the vector by a scalar.

##### Parameters

<i>scalar</i>	The scalar to multiply by.
---------------	----------------------------

##### Returns

The product of the vector and the scalar.

#### 4.36.1.10 operator\*() [2/2]

```
Vector rtr::Vector::operator* (
    const Vector & other ) const [inline]
```

Multiplies two vectors.

**Parameters**

<i>other</i>	The other vector to multiply.
--------------	-------------------------------

**Returns**

The product of the two vectors.

**4.36.1.11 operator+() [1/2]**

```
Vector rtr::Vector::operator+ (  
    const double scalar ) const [inline]
```

Adds a scalar to the vector.

**Parameters**

<i>scalar</i>	The scalar to add.
---------------	--------------------

**Returns**

The sum of the vector and the scalar.

**4.36.1.12 operator+() [2/2]**

```
Vector rtr::Vector::operator+ (  
    const Vector & other ) const [inline]
```

Adds two vectors.

**Parameters**

<i>other</i>	The other vector to add.
--------------	--------------------------

**Returns**

The sum of the two vectors.

**4.36.1.13 operator-()**

```
Vector rtr::Vector::operator- (  
    const Vector & other ) const [inline]
```

Subtracts two vectors.

## Parameters

<i>other</i>	The other vector to subtract.
--------------	-------------------------------

## Returns

The difference of the two vectors.

**4.36.1.14 operator/()**

```
Vector rtr::Vector::operator/ (
    const double scalar ) const [inline]
```

Divides the vector by a scalar.

## Parameters

<i>scalar</i>	The scalar to divide by.
---------------	--------------------------

## Returns

The quotient of the vector and the scalar.

**4.36.1.15 setVector() [1/2]**

```
void rtr::Vector::setVector (
    const double x,
    const double y,
    const double z ) [inline]
```

Sets the vector to the given *x*, *y*, and *z* values.

## Parameters

<i>x</i>	The x-component of the vector.
<i>y</i>	The y-component of the vector.
<i>z</i>	The z-component of the vector.

**4.36.1.16 setVector() [2/2]**

```
void rtr::Vector::setVector (
    const vector_t & position ) [inline]
```

Sets the vector to the given vector struct.

#### Parameters

<i>position</i>	The vector struct to set.
-----------------	---------------------------

#### 4.36.1.17 setX()

```
void rtr::Vector::setX (
    const double x ) [inline]
```

Sets the x-component of the vector.

#### Parameters

<i>x</i>	The x-component to set.
----------	-------------------------

#### 4.36.1.18 setY()

```
void rtr::Vector::setY (
    const double y ) [inline]
```

Sets the y-component of the vector.

#### Parameters

<i>y</i>	The y-component to set.
----------	-------------------------

#### 4.36.1.19 setZ()

```
void rtr::Vector::setZ (
    const double z ) [inline]
```

Sets the z-component of the vector.

#### Parameters

<i>z</i>	The z-component to set.
----------	-------------------------

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

- App/include/RayTracer/Utils/[Vector.hpp](#)

## 4.37 vector\_s Struct Reference

A struct representing a 3D vector.

```
#include <Vector.hpp>
```

### 4.37.1 Detailed Description

A struct representing a 3D vector.

Type alias for a 3D vector.

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

- App/include/RayTracer/Utils/[Vector.hpp](#)





## Chapter 5

# File Documentation

### 5.1 App/include/RayTracer/Abstraction/ALight.hpp File Reference

```
#include "RayTracer/Abstraction/ILight.hpp"
```

#### Classes

- class [rtr::ALight](#)  
*An abstract class for lights.*

### 5.2 App/include/RayTracer/Abstraction/AMaterial.hpp File Reference

```
#include "RayTracer/Abstraction/IMaterial.hpp"
```

#### Classes

- class [rtr::AMaterial](#)  
*An abstract class for materials, based on the interface [IMaterials](#).*

### 5.3 App/include/RayTracer/Abstraction/ARenderer.hpp File Reference

```
#include "RayTracer/Abstraction/IRenderer.hpp"
```

#### Classes

- class [rtr::ARenderer](#)  
*An abstract class for renderers, based on the interface [IRenderer](#).*

## 5.4 App/include/RayTracer/Abstraction/AShape.hpp File Reference

```
#include "RayTracer/Abstraction/IShape.hpp"
```

### Classes

- class [rtr::AShape](#)  
*An abstract class for shapes, based on the interface [IShape](#).*

## 5.5 App/include/RayTracer/Abstraction/ILight.hpp File Reference

```
#include <vector>
#include "RayTracer/Abstraction/IPlugin.hpp"
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Constants.hpp"
#include "RayTracer/Utils/Vector.hpp"
#include "RayTracer/Utils/Color.hpp"
```

### Classes

- class [rtr::ILight](#)  
*An interface for lights.*

## 5.6 App/include/RayTracer/Abstraction/IMaterial.hpp File Reference

```
#include "RayTracer/Abstraction/IPlugin.hpp"
#include "RayTracer/Utils/Color.hpp"
#include "RayTracer/Constants.hpp"
```

### Classes

- class [rtr::IMaterial](#)  
*An interface for materials.*

## 5.7 App/include/RayTracer/Abstraction/IPlugin.hpp File Reference

```
#include <string>
```

## Classes

- class [rtr::IPlugin](#)  
*An interface for plugins.*

## 5.8 App/include/RayTracer/Abstraction/IRenderer.hpp File Reference

```
#include <vector>
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Abstraction/ALight.hpp"
#include "RayTracer/Scene/Camera.hpp"
#include "RayTracer/Utils/Resolution.hpp"
#include "RayTracer/Constants.hpp"
```

## Classes

- class [rtr::IRenderer](#)  
*An interface for renderers.*

## 5.9 App/include/RayTracer/Abstraction/IShape.hpp File Reference

```
#include <memory>
#include "RayTracer/Abstraction/AMaterial.hpp"
#include "RayTracer/Constants.hpp"
#include "RayTracer/Utils/Vector.hpp"
#include "RayTracer/Utils/RayHit.hpp"
```

## Classes

- class [rtr::IShape](#)  
*An interface used to get the shape's parameters based on the configuration file.*

## 5.10 App/include/RayTracer/Composite/Material.hpp File Reference

```
#include <vector>
#include <memory>
#include "RayTracer/Abstraction/AMaterial.hpp"
```

## Classes

- class [rtr::CompositeMaterial](#)  
*A class to create a composite material.*

## 5.11 App/include/RayTracer/Factory/Material.hpp File Reference

```
#include "RayTracer/Abstraction/AMaterial.hpp"  
#include "RayTracer/Loader/Plugin.hpp"
```

### Classes

- class [rtr::MaterialFactory](#)  
*A factory class for the materials of the shapes.*

## 5.12 App/include/RayTracer/Constants.hpp File Reference

### Enumerations

- enum class **RendererType** { PPM , SFML , NONE }
- enum class **ShapeType** {  
    **SPHERE** , **PLANE** , **CYLINDER** , **CONE** ,  
    **NONE** }
- enum class **LightType** { **AMBIENT** , **DIRECTIONAL** , **POINT** , **NONE** }
- enum class **MaterialType** { **TRANSPARENT** , **REFLECTIVE** , **NONE** }

## 5.13 App/include/RayTracer/Core.hpp File Reference

```
#include "RayTracer/Abstraction/ARenderer.hpp"  
#include "RayTracer/Scene/Scene.hpp"
```

### Classes

- class [rtr::Core](#)  
*A class representing the core functionality of the ray tracer.*
- class [rtr::Core::CoreException](#)  
*An exception class for core errors.*

## 5.14 App/include/RayTracer/Factory/Light.hpp File Reference

```
#include "RayTracer/Abstraction/ALight.hpp"  
#include "RayTracer/Loader/Plugin.hpp"
```

### Classes

- class [rtr::LightFactory](#)  
*A factory class for the lights.*

## 5.15 App/include/RayTracer/Factory/Rendererer.hpp File Reference

```
#include "RayTracer/Abstraction/ARenderer.hpp"
#include "RayTracer/Loader/Plugin.hpp"
```

### Classes

- class [rtr::RendererFactory](#)  
*A factory class to create the renderers.*

## 5.16 App/include/RayTracer/Factory/Shape.hpp File Reference

```
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Loader/Plugin.hpp"
```

### Classes

- class [rtr::ShapeFactory](#)  
*A factory class for the shapes.*

## 5.17 App/include/RayTracer/Loader/Plugin.hpp File Reference

```
#include <dlfcn.h>
#include <unordered_map>
#include <filesystem>
#include "RayTracer/Abstraction/IRenderer.hpp"
#include "RayTracer/Exception/RunTime.hpp"
```

### Classes

- class [rtr::PluginLoader](#)  
*A class to load the plugins.*

## 5.18 App/include/RayTracer/Parser.hpp File Reference

```
#include <iostream>
#include <libconfig.h++>
#include "RayTracer/Scene/Scene.hpp"
```

## Classes

- class [rtr::Parser](#)  
*Class dedicated to the parsing of configuration files and command-line arguments.*
- class [rtr::Parser::ParserException](#)  
*Exception class for errors in the parsers.*

## 5.19 App/include/RayTracer/Scene/Camera.hpp File Reference

```
#include "RayTracer/Utils/Vector.hpp"
```

## Classes

- class [rtr::Camera](#)  
*A class to handle the camera.*

## 5.20 App/include/RayTracer/Scene/Scene.hpp File Reference

```
#include <vector>
#include "RayTracer/Scene/Camera.hpp"
#include "RayTracer/Abstraction/ALight.hpp"
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Factory/Renderer.hpp"
```

## Classes

- class [rtr::Scene](#)  
*A class to represent the scene.*

## 5.21 App/include/RayTracer/Utils/Color.hpp File Reference

```
#include <cstdint>
```

## Classes

- class [rtr::Color](#)  
*Class representing RGB colors.*

## Typedefs

- using [rtr::color\\_t](#) = struct [color\\_s](#) { uint8\_t r{0}

## Variables

- `uint8_t rtr::g {0}`  
*Green color component.*
- `uint8_t rtr::b {0}`  
*Blue color component.*

## 5.22 App/include/RayTracer/Utils/RayHit.hpp File Reference

```
#include "RayTracer/Utils/Vector.hpp"
```

## Classes

- class `rtr::RayHit`  
*A class representing a ray hit in 3D space.*

## Typedefs

- using `rtr::ray_hit_t` = struct `ray_hit_s` { Vector point

## Variables

- Vector `rtr::normal`  
*The normal vector at the point of intersection.*
- double `rtr::distance`  
*The distance from the ray origin to the point of intersection.*

## 5.23 App/include/RayTracer/Utils/Resolution.hpp File Reference

```
#include <cstdint>
```

## Classes

- class `rtr::Resolution`  
*Class representing the resolution of an image.*

## Typedefs

- using `rtr::resolution_t` = struct `resolution_s` { uint16\_t width

## Variables

- `uint16_t rtr::height`  
*The height of the image.*

## 5.24 App/include/RayTracer/Utils/Vector.hpp File Reference

```
#include <stdint>
#include <cmath>
```

## Classes

- class `rtr::Vector`

## Typedefs

- using `rtr::vector_t` = struct `vector_s` { double x

## Variables

- double `rtr::y`  
*The y-component of the vector.*
- double `rtr::z`  
*The z-component of the vector.*



# Index

- addLight
  - rtr::Scene, [64](#)
- addMaterial
  - rtr::CompositeMaterial, [26](#)
- addShape
  - rtr::Scene, [64](#)
- App/include/RayTracer/Abstraction/ALight.hpp, [79](#)
- App/include/RayTracer/Abstraction/AMaterial.hpp, [79](#)
- App/include/RayTracer/Abstraction/ARenderer.hpp, [79](#)
- App/include/RayTracer/Abstraction/AShape.hpp, [80](#)
- App/include/RayTracer/Abstraction/ILight.hpp, [80](#)
- App/include/RayTracer/Abstraction/IMaterial.hpp, [80](#)
- App/include/RayTracer/Abstraction/IPlugin.hpp, [80](#)
- App/include/RayTracer/Abstraction/IRenderer.hpp, [81](#)
- App/include/RayTracer/Abstraction/IShape.hpp, [81](#)
- App/include/RayTracer/Composite/Material.hpp, [81](#)
- App/include/RayTracer/Constants.hpp, [82](#)
- App/include/RayTracer/Core.hpp, [82](#)
- App/include/RayTracer/Factory/Light.hpp, [82](#)
- App/include/RayTracer/Factory/Material.hpp, [82](#)
- App/include/RayTracer/Factory/Renderer.hpp, [83](#)
- App/include/RayTracer/Factory/Shape.hpp, [83](#)
- App/include/RayTracer/Loader/Plugin.hpp, [83](#)
- App/include/RayTracer/Parser.hpp, [83](#)
- App/include/RayTracer/Scene/Camera.hpp, [84](#)
- App/include/RayTracer/Scene/Scene.hpp, [84](#)
- App/include/RayTracer/Utils/Color.hpp, [84](#)
- App/include/RayTracer/Utils/RayHit.hpp, [85](#)
- App/include/RayTracer/Utils/Resolution.hpp, [85](#)
- App/include/RayTracer/Utils/Vector.hpp, [86](#)
- color\_s, [25](#)
- convertInt
  - rtr::Parser, [46](#)
- createLight
  - rtr::LightFactory, [43](#)
- createMaterial
  - rtr::MaterialFactory, [44](#)
- createRenderer
  - rtr::RendererFactory, [59](#)
- createShape
  - rtr::ShapeFactory, [68](#), [69](#)
- cross
  - rtr::Vector, [71](#)
- dot
  - rtr::Vector, [71](#)
- getBackgroundColor
  - rtr::ARenderer, [12](#)
- getDistance
  - rtr::AShape, [15](#)
  - rtr::IShape, [38](#)
- getFov
  - rtr::Camera, [19](#)
- getHeight
  - rtr::AShape, [16](#)
  - rtr::IShape, [39](#)
  - rtr::Resolution, [61](#)
- getInstance
  - rtr::PluginLoader, [51](#)
- getLights
  - rtr::Scene, [65](#)
- getMaterial
  - rtr::AShape, [16](#)
  - rtr::IShape, [39](#)
- getName
  - rtr::ARenderer, [12](#)
  - rtr::IRenderer, [35](#)
- getNormal
  - rtr::AShape, [16](#)
  - rtr::IShape, [39](#)
- getPixels
  - rtr::ARenderer, [12](#)
  - rtr::IRenderer, [35](#)
- getPlugin
  - rtr::PluginLoader, [51](#)
- getPluginName
  - rtr::Ambient, [10](#)
  - rtr::CompositeMaterial, [26](#)
  - rtr::Directional, [29](#)
  - rtr::IPlugin, [33](#)
  - rtr::Point, [52](#)
  - rtr::PPM, [55](#)
  - rtr::Reflective, [58](#)
  - rtr::SFML, [66](#)
  - rtr::Transparent, [70](#)
- getPosition
  - rtr::AShape, [16](#)
  - rtr::IShape, [40](#)
- getRadius
  - rtr::AShape, [17](#)
  - rtr::IShape, [40](#)
- getRenderer
  - rtr::Scene, [65](#)
- getResolution
  - rtr::ARenderer, [13](#)
  - rtr::IRenderer, [35](#)

- getRotation
  - rtr::AShape, 17
  - rtr::IShape, 40
- getShapes
  - rtr::Scene, 65
- getType
  - rtr::ARenderer, 13
  - rtr::AShape, 17
  - rtr::IRenderer, 36
  - rtr::IShape, 40
- getValue
  - rtr::Resolution, 61
  - rtr::Vector, 72
- getVector
  - rtr::Parser, 46
- getWidth
  - rtr::Resolution, 61
- getX
  - rtr::Vector, 72
- getY
  - rtr::Vector, 72
- getZ
  - rtr::Vector, 72
- hits
  - rtr::IShape, 41
- length
  - rtr::Vector, 72
- LightColor
  - rtr::Ambient, 10
  - rtr::Directional, 29
  - rtr::ILight, 30
  - rtr::Point, 52
- normalize
  - rtr::Vector, 73
- operator\*
  - rtr::Color, 21, 22
  - rtr::Vector, 73
- operator\*=
  - rtr::Color, 22
- operator+
  - rtr::Color, 22
  - rtr::Vector, 74
- operator+=
  - rtr::Color, 24
- operator-
  - rtr::Vector, 74
- operator/
  - rtr::Vector, 75
- parseArgs
  - rtr::Parser, 47
- parseCamera
  - rtr::Parser, 47
- parseFile
  - rtr::Parser, 47
- parseLights
  - rtr::Parser, 48
- parseLightType
  - rtr::Parser, 48
- parseMaterial
  - rtr::Parser, 48
- parseRenderer
  - rtr::Parser, 49
- parseShapes
  - rtr::Parser, 49
- parseShapeType
  - rtr::Parser, 49
- ray
  - rtr::Camera, 19
- ray\_hit\_s, 55
- render
  - rtr::IRenderer, 36
  - rtr::PPM, 55
  - rtr::SFML, 67
- resolution\_s, 63
- rtr::ALight, 7
  - setType, 8
- rtr::AMaterial, 8
  - setReflectivity, 9
  - setTransparency, 9
- rtr::Ambient, 10
  - getPluginName, 10
  - LightColor, 10
- rtr::ARenderer, 11
  - getBackgroundColor, 12
  - getName, 12
  - getPixels, 12
  - getResolution, 13
  - getType, 13
  - setName, 13
  - setPixels, 14
  - setType, 14
- rtr::AShape, 14
  - getDistance, 15
  - getHeight, 16
  - getMaterial, 16
  - getNormal, 16
  - getPosition, 16
  - getRadius, 17
  - getRotation, 17
  - getType, 17
  - setHeight, 17
  - setMaterial, 18
  - setRadius, 18
  - setType, 18
- rtr::Camera, 19
  - getFov, 19
  - ray, 19
  - setFov, 20
- rtr::Color, 20
  - operator\*, 21, 22
  - operator\*=, 22
  - operator+, 22

- operator+=, 24
- setColor, 24
- rtr::CompositeMaterial, 25
  - addMaterial, 26
  - getPluginName, 26
- rtr::Core, 27
  - runRayTracer, 27
- rtr::Core::CoreException, 27
- rtr::Directional, 28
  - getPluginName, 29
  - LightColor, 29
- rtr::ILight, 29
  - LightColor, 30
  - setType, 31
- rtr::IMaterial, 31
  - setReflectivity, 32
  - setTransparency, 32
- rtr::IPlugin, 33
  - getPluginName, 33
- rtr::IRenderer, 34
  - getBackgroundColor, 35
  - getName, 35
  - getPixels, 35
  - getResolution, 35
  - getType, 36
  - render, 36
  - setName, 36
  - setPixels, 37
  - setType, 37
- rtr::IShape, 37
  - getDistance, 38
  - getHeight, 39
  - getMaterial, 39
  - getNormal, 39
  - getPosition, 40
  - getRadius, 40
  - getRotation, 40
  - getType, 40
  - hits, 41
  - setHeight, 41
  - setMaterial, 41
  - setRadius, 42
  - setType, 42
- rtr::LightFactory, 43
  - createLight, 43
- rtr::MaterialFactory, 44
  - createMaterial, 44
- rtr::Parser, 45
  - convertInt, 46
  - getVector, 46
  - parseArgs, 47
  - parseCamera, 47
  - parseFile, 47
  - parseLights, 48
  - parseLightType, 48
  - parseMaterial, 48
  - parseRenderer, 49
  - parseShapes, 49
  - parseShapeType, 49
- rtr::Parser::ParserException, 50
- rtr::PluginLoader, 51
  - getInstance, 51
  - getPlugin, 51
- rtr::Point, 52
  - getPluginName, 52
  - LightColor, 52
- rtr::PPM, 54
  - getPluginName, 55
  - render, 55
- rtr::RayHit, 56
  - setDistance, 56
  - setNormal, 57
  - setPoint, 57
  - setRayHit, 57
- rtr::Reflective, 58
  - getPluginName, 58
- rtr::RendererFactory, 59
  - createRenderer, 59
- rtr::Resolution, 60
  - getHeight, 61
  - getValue, 61
  - getWidth, 61
  - setHeight, 61
  - setResolution, 62
  - setWidth, 62
- rtr::RunTimeException, 63
- rtr::Scene, 63
  - addLight, 64
  - addShape, 64
  - getLights, 65
  - getRenderer, 65
  - getShapes, 65
  - setCamera, 65
  - setRenderer, 66
- rtr::SFML, 66
  - getPluginName, 66
  - render, 67
- rtr::ShapeFactory, 67
  - createShape, 68, 69
- rtr::Transparent, 69
  - getPluginName, 70
- rtr::Vector, 70
  - cross, 71
  - dot, 71
  - getValue, 72
  - getX, 72
  - getY, 72
  - getZ, 72
  - length, 72
  - normalize, 73
  - operator\*, 73
  - operator+, 74
  - operator-, 74
  - operator/, 75
  - setVector, 75
  - setX, 76

- setY, [76](#)
- setZ, [76](#)
- runRayTracer
  - rtr::Core, [27](#)
- setCamera
  - rtr::Scene, [65](#)
- setColor
  - rtr::Color, [24](#)
- setDistance
  - rtr::RayHit, [56](#)
- setFov
  - rtr::Camera, [20](#)
- setHeight
  - rtr::AShape, [17](#)
  - rtr::IShape, [41](#)
  - rtr::Resolution, [61](#)
- setMaterial
  - rtr::AShape, [18](#)
  - rtr::IShape, [41](#)
- setName
  - rtr::ARenderer, [13](#)
  - rtr::IRenderer, [36](#)
- setNormal
  - rtr::RayHit, [57](#)
- setPixels
  - rtr::ARenderer, [14](#)
  - rtr::IRenderer, [37](#)
- setPoint
  - rtr::RayHit, [57](#)
- setRadius
  - rtr::AShape, [18](#)
  - rtr::IShape, [42](#)
- setRayHit
  - rtr::RayHit, [57](#)
- setReflectivity
  - rtr::AMaterial, [9](#)
  - rtr::IMaterial, [32](#)
- setRenderer
  - rtr::Scene, [66](#)
- setResolution
  - rtr::Resolution, [62](#)
- setTransparency
  - rtr::AMaterial, [9](#)
  - rtr::IMaterial, [32](#)
- setType
  - rtr::ALight, [8](#)
  - rtr::ARenderer, [14](#)
  - rtr::AShape, [18](#)
  - rtr::ILight, [31](#)
  - rtr::IRenderer, [37](#)
  - rtr::IShape, [42](#)
- setVector
  - rtr::Vector, [75](#)
- setWidth
  - rtr::Resolution, [62](#)
- setX
  - rtr::Vector, [76](#)
- setY
  - rtr::Vector, [76](#)
- setZ
  - rtr::Vector, [76](#)
- vector\_s, [77](#)