SFML

* [Main Page](http://docs.google.com/index.htm)
* [Modules](http://docs.google.com/modules.htm)
* [Classes](http://docs.google.com/annotated.htm)
* [Files](http://docs.google.com/files.htm)
* [Class List](http://docs.google.com/annotated.htm)
* [Class Index](http://docs.google.com/classes.htm)
* [Class Hierarchy](http://docs.google.com/hierarchy.htm)
* [Class Members](http://docs.google.com/functions.htm)
* **sf**
* [Shader](http://docs.google.com/classsf_1_1Shader.htm)

[Classes](#_gjdgxs) | [Public Types](#_30j0zll) | [Public Member Functions](#_1fob9te) | [Static Public Member Functions](#_3znysh7) | [Static Public Attributes](#_2et92p0) | [Static Private Member Functions](#_tyjcwt) | [List of all members](http://docs.google.com/classsf_1_1Shader-members.htm)

sf::Shader Class Reference

[Graphics module](http://docs.google.com/group__graphics.htm)

[Shader](http://docs.google.com/classsf_1_1Shader.htm) class (vertex and fragment) [More...](http://docs.google.com/classsf_1_1Shader.htm#details)

#include <[Shader.hpp](http://docs.google.com/Shader_8hpp_source.htm)>

Inheritance diagram for sf::Shader:



| Classes | |
| --- | --- |
| struct | [CurrentTextureType](http://docs.google.com/structsf_1_1Shader_1_1CurrentTextureType.htm) |
|  | Special type/value that can be passed to setParameter, and that represents the texture of the object being drawn. [More...](http://docs.google.com/structsf_1_1Shader_1_1CurrentTextureType.htm#details) |
|  | |

| Public Types | |
| --- | --- |
| enum | [Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) {  [Vertex](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3a8718008f827eb32e29bbdd1791c62dce),  [Fragment](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3ace6e88eec3a56b2e55ee3c8e64e9b89a)  } |
|  | Types of shaders. [More...](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) |
|  | |

| Public Member Functions | |
| --- | --- |
|  | [Shader](http://docs.google.com/classsf_1_1Shader.htm#a1d7f28f26b4122959fcafec871c2c3c5) () |
|  | Default constructor. |
|  | |
|  | [~Shader](http://docs.google.com/classsf_1_1Shader.htm#a4bac6cc8b046ecd8fb967c145a2380e6) () |
|  | Destructor. |
|  | |
| bool | [loadFromFile](http://docs.google.com/classsf_1_1Shader.htm#a053a5632848ebaca2fcd8ba29abe9e6e) (const std::string &filename, [Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) type) |
|  | Load either the vertex or fragment shader from a file. |
|  | |
| bool | [loadFromFile](http://docs.google.com/classsf_1_1Shader.htm#ac9d7289966fcef562eeb92271c03e3dc) (const std::string &vertexShaderFilename, const std::string &fragmentShaderFilename) |
|  | Load both the vertex and fragment shaders from files. |
|  | |
| bool | [loadFromMemory](http://docs.google.com/classsf_1_1Shader.htm#ac92d46bf71dff2d791117e4e472148aa) (const std::string &shader, [Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) type) |
|  | Load either the vertex or fragment shader from a source code in memory. |
|  | |
| bool | [loadFromMemory](http://docs.google.com/classsf_1_1Shader.htm#ae34e94070d7547a890166b7993658a9b) (const std::string &vertexShader, const std::string &fragmentShader) |
|  | Load both the vertex and fragment shaders from source codes in memory. |
|  | |
| bool | [loadFromStream](http://docs.google.com/classsf_1_1Shader.htm#a2ee1b130c0606e4f8bcdf65c1efc2a53) ([InputStream](http://docs.google.com/classsf_1_1InputStream.htm) &stream, [Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) type) |
|  | Load either the vertex or fragment shader from a custom stream. |
|  | |
| bool | [loadFromStream](http://docs.google.com/classsf_1_1Shader.htm#a3b7958159ffb5596c4babc3052e35465) ([InputStream](http://docs.google.com/classsf_1_1InputStream.htm) &vertexShaderStream, [InputStream](http://docs.google.com/classsf_1_1InputStream.htm) &fragmentShaderStream) |
|  | Load both the vertex and fragment shaders from custom streams. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#a47e4dd78f0752ae08664b4ee616db1cf) (const std::string &name, float x) |
|  | Change a float parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#ab8d379f40810b8e3eadebee81aedd231) (const std::string &name, float x, float y) |
|  | Change a 2-components vector parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#a7e36e044d6b8adca8339f40c5a4b1801) (const std::string &name, float x, float y, float z) |
|  | Change a 3-components vector parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#aeb468f1bc2d26750b96b74f1e19027fb) (const std::string &name, float x, float y, float z, float w) |
|  | Change a 4-components vector parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#a3ac473ece2c6fa26dc5032c07fd7288e) (const std::string &name, const [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) &vector) |
|  | Change a 2-components vector parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#a87d4a0c6dc70ae68aecc0dda3f343c07) (const std::string &name, const [Vector3f](http://docs.google.com/classsf_1_1Vector3.htm) &vector) |
|  | Change a 3-components vector parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#aa8618119ed4399df3fd33e78ee96b4fc) (const std::string &name, const [Color](http://docs.google.com/classsf_1_1Color.htm) &color) |
|  | Change a color parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#a39c387cc30e249b22a0c478703b8cc9a) (const std::string &name, const [sf::Transform](http://docs.google.com/classsf_1_1Transform.htm) &transform) |
|  | Change a matrix parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#a7f58ab5c0a1084f238dfcec86602daa1) (const std::string &name, const [Texture](http://docs.google.com/classsf_1_1Texture.htm) &texture) |
|  | Change a texture parameter of the shader. |
|  | |
| void | [setParameter](http://docs.google.com/classsf_1_1Shader.htm#af06b4cba0bab915fa01032b063909044) (const std::string &name, [CurrentTextureType](http://docs.google.com/structsf_1_1Shader_1_1CurrentTextureType.htm)) |
|  | Change a texture parameter of the shader. |
|  | |

| Static Public Member Functions | |
| --- | --- |
| static void | [bind](http://docs.google.com/classsf_1_1Shader.htm#a09778f78afcbeb854d608c8dacd8ea30) (const [Shader](http://docs.google.com/classsf_1_1Shader.htm) \*shader) |
|  | Bind a shader for rendering. |
|  | |
| static bool | [isAvailable](http://docs.google.com/classsf_1_1Shader.htm#ad22474690bafe4a305c1b9826b1bd86a) () |
|  | Tell whether or not the system supports shaders. |
|  | |

| Static Public Attributes | |
| --- | --- |
| static [CurrentTextureType](http://docs.google.com/structsf_1_1Shader_1_1CurrentTextureType.htm) | **CurrentTexture** |
|  | |

| Static Private Member Functions | |
| --- | --- |
| static void | [ensureGlContext](http://docs.google.com/classsf_1_1GlResource.htm#ae0efa7935241644608ca32ba47b22a33) () |
|  | Make sure that a valid OpenGL context exists in the current thread. |
|  | |

## Detailed Description

[Shader](http://docs.google.com/classsf_1_1Shader.htm) class (vertex and fragment)

Shaders are programs written using a specific language, executed directly by the graphics card and allowing to apply real-time operations to the rendered entities.

There are two kinds of shaders:

* [Vertex](http://docs.google.com/classsf_1_1Vertex.htm) shaders, that process vertices
* Fragment (pixel) shaders, that process pixels

A [sf::Shader](http://docs.google.com/classsf_1_1Shader.htm) can be composed of either a vertex shader alone, a fragment shader alone, or both combined (see the variants of the load functions).

Shaders are written in GLSL, which is a C-like language dedicated to OpenGL shaders. You'll probably need to learn its basics before writing your own shaders for SFML.

Like any C/C++ program, a shader has its own variables that you can set from your C++ application. [sf::Shader](http://docs.google.com/classsf_1_1Shader.htm) handles 5 different types of variables:

* floats
* vectors (2, 3 or 4 components)
* colors
* textures
* transforms (matrices)

The value of the variables can be changed at any time with the various overloads of the setParameter function:

shader.setParameter("offset", 2.f);

shader.setParameter("point", 0.5f, 0.8f, 0.3f);

shader.setParameter("color", [sf::Color](http://docs.google.com/classsf_1_1Color.htm)(128, 50, 255));

shader.setParameter("matrix", transform); // transform is a sf::Transform

shader.setParameter("overlay", texture); // texture is a sf::Texture

shader.setParameter("texture", sf::Shader::CurrentTexture);

The special Shader::CurrentTexture argument maps the given texture variable to the current texture of the object being drawn (which cannot be known in advance).

To apply a shader to a drawable, you must pass it as an additional parameter to the Draw function:

window.draw(sprite, &shader);

... which is in fact just a shortcut for this:

[sf::RenderStates](http://docs.google.com/classsf_1_1RenderStates.htm) states;

states.[shader](http://docs.google.com/classsf_1_1RenderStates.htm#ad4f79ecdd0c60ed0d24fbe555b221bd8) = &shader;

window.draw(sprite, states);

In the code above we pass a pointer to the shader, because it may be null (which means "no shader").

Shaders can be used on any drawable, but some combinations are not interesting. For example, using a vertex shader on a [sf::Sprite](http://docs.google.com/classsf_1_1Sprite.htm) is limited because there are only 4 vertices, the sprite would have to be subdivided in order to apply wave effects. Another bad example is a fragment shader with [sf::Text](http://docs.google.com/classsf_1_1Text.htm): the texture of the text is not the actual text that you see on screen, it is a big texture containing all the characters of the font in an arbitrary order; thus, texture lookups on pixels other than the current one may not give you the expected result.

Shaders can also be used to apply global post-effects to the current contents of the target (like the old sf::PostFx class in SFML 1). This can be done in two different ways:

* draw everything to a [sf::RenderTexture](http://docs.google.com/classsf_1_1RenderTexture.htm), then draw it to the main target using the shader
* draw everything directly to the main target, then use sf::Texture::update(Window&) to copy its contents to a texture and draw it to the main target using the shader

The first technique is more optimized because it doesn't involve retrieving the target's pixels to system memory, but the second one doesn't impact the rendering process and can be easily inserted anywhere without impacting all the code.

Like [sf::Texture](http://docs.google.com/classsf_1_1Texture.htm) that can be used as a raw OpenGL texture, [sf::Shader](http://docs.google.com/classsf_1_1Shader.htm) can also be used directly as a raw shader for custom OpenGL geometry.

[sf::Shader::bind](http://docs.google.com/classsf_1_1Shader.htm#a09778f78afcbeb854d608c8dacd8ea30)(&shader);

... render OpenGL geometry ...

sf::Shader::bind(NULL);

Definition at line [51](http://docs.google.com/Shader_8hpp_source.htm#l00051) of file [Shader.hpp](http://docs.google.com/Shader_8hpp_source.htm).

## Member Enumeration Documentation

| enum [sf::Shader::Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) |
| --- |

Types of shaders.

**Enumerator:**

| *Vertex* | [Vertex](http://docs.google.com/classsf_1_1Vertex.htm) shader. |
| --- | --- |
| *Fragment* | Fragment (pixel) shader. |

Definition at line [59](http://docs.google.com/Shader_8hpp_source.htm#l00059) of file [Shader.hpp](http://docs.google.com/Shader_8hpp_source.htm).

## Constructor & Destructor Documentation

| sf::Shader::Shader | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Default constructor.

This constructor creates an invalid shader.

| sf::Shader::~Shader | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Destructor.

## Member Function Documentation

| | static void sf::Shader::bind | ( | const [Shader](http://docs.google.com/classsf_1_1Shader.htm) \* | *shader* | ) |  | | --- | --- | --- | --- | --- | --- | | static |
| --- | --- | --- | --- | --- | --- | --- | --- |

Bind a shader for rendering.

This function is not part of the graphics API, it mustn't be used when drawing SFML entities. It must be used only if you mix [sf::Shader](http://docs.google.com/classsf_1_1Shader.htm) with OpenGL code.

[sf::Shader](http://docs.google.com/classsf_1_1Shader.htm) s1, s2;

...

sf::Shader::bind(&s1);

// draw OpenGL stuff that use s1...

[sf::Shader::bind](http://docs.google.com/classsf_1_1Shader.htm#a09778f78afcbeb854d608c8dacd8ea30)(&s2);

// draw OpenGL stuff that use s2...

[sf::Shader::bind](http://docs.google.com/classsf_1_1Shader.htm#a09778f78afcbeb854d608c8dacd8ea30)(NULL);

// draw OpenGL stuff that use no shader...

Parameters

| shader | [Shader](http://docs.google.com/classsf_1_1Shader.htm) to bind, can be null to use no shader |
| --- | --- |

| | static bool sf::Shader::isAvailable | ( |  | ) |  | | --- | --- | --- | --- | --- | | static |
| --- | --- | --- | --- | --- | --- | --- |

Tell whether or not the system supports shaders.

This function should always be called before using the shader features. If it returns false, then any attempt to use [sf::Shader](http://docs.google.com/classsf_1_1Shader.htm) will fail.

ReturnsTrue if shaders are supported, false otherwise

| bool sf::Shader::loadFromFile | ( | const std::string & | *filename*, |
| --- | --- | --- | --- |
|  |  | [Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) | *type* |
|  | ) |  |  |

Load either the vertex or fragment shader from a file.

This function loads a single shader, either vertex or fragment, identified by the second argument. The source must be a text file containing a valid shader in GLSL language. GLSL is a C-like language dedicated to OpenGL shaders; you'll probably need to read a good documentation for it before writing your own shaders.

Parameters

| filename | Path of the vertex or fragment shader file to load |
| --- | --- |
| type | Type of shader (vertex or fragment) |

ReturnsTrue if loading succeeded, false if it failed See Also[loadFromMemory](http://docs.google.com/classsf_1_1Shader.htm#ac92d46bf71dff2d791117e4e472148aa), [loadFromStream](http://docs.google.com/classsf_1_1Shader.htm#a2ee1b130c0606e4f8bcdf65c1efc2a53)

| bool sf::Shader::loadFromFile | ( | const std::string & | *vertexShaderFilename*, |
| --- | --- | --- | --- |
|  |  | const std::string & | *fragmentShaderFilename* |
|  | ) |  |  |

Load both the vertex and fragment shaders from files.

This function loads both the vertex and the fragment shaders. If one of them fails to load, the shader is left empty (the valid shader is unloaded). The sources must be text files containing valid shaders in GLSL language. GLSL is a C-like language dedicated to OpenGL shaders; you'll probably need to read a good documentation for it before writing your own shaders.

Parameters

| vertexShaderFilename | Path of the vertex shader file to load |
| --- | --- |
| fragmentShaderFilename | Path of the fragment shader file to load |

ReturnsTrue if loading succeeded, false if it failed See Also[loadFromMemory](http://docs.google.com/classsf_1_1Shader.htm#ac92d46bf71dff2d791117e4e472148aa), [loadFromStream](http://docs.google.com/classsf_1_1Shader.htm#a2ee1b130c0606e4f8bcdf65c1efc2a53)

| bool sf::Shader::loadFromMemory | ( | const std::string & | *shader*, |
| --- | --- | --- | --- |
|  |  | [Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) | *type* |
|  | ) |  |  |

Load either the vertex or fragment shader from a source code in memory.

This function loads a single shader, either vertex or fragment, identified by the second argument. The source code must be a valid shader in GLSL language. GLSL is a C-like language dedicated to OpenGL shaders; you'll probably need to read a good documentation for it before writing your own shaders.

Parameters

| shader | [String](http://docs.google.com/classsf_1_1String.htm) containing the source code of the shader |
| --- | --- |
| type | Type of shader (vertex or fragment) |

ReturnsTrue if loading succeeded, false if it failed See Also[loadFromFile](http://docs.google.com/classsf_1_1Shader.htm#a053a5632848ebaca2fcd8ba29abe9e6e), [loadFromStream](http://docs.google.com/classsf_1_1Shader.htm#a2ee1b130c0606e4f8bcdf65c1efc2a53)

| bool sf::Shader::loadFromMemory | ( | const std::string & | *vertexShader*, |
| --- | --- | --- | --- |
|  |  | const std::string & | *fragmentShader* |
|  | ) |  |  |

Load both the vertex and fragment shaders from source codes in memory.

This function loads both the vertex and the fragment shaders. If one of them fails to load, the shader is left empty (the valid shader is unloaded). The sources must be valid shaders in GLSL language. GLSL is a C-like language dedicated to OpenGL shaders; you'll probably need to read a good documentation for it before writing your own shaders.

Parameters

| vertexShader | [String](http://docs.google.com/classsf_1_1String.htm) containing the source code of the vertex shader |
| --- | --- |
| fragmentShader | [String](http://docs.google.com/classsf_1_1String.htm) containing the source code of the fragment shader |

ReturnsTrue if loading succeeded, false if it failed See Also[loadFromFile](http://docs.google.com/classsf_1_1Shader.htm#a053a5632848ebaca2fcd8ba29abe9e6e), [loadFromStream](http://docs.google.com/classsf_1_1Shader.htm#a2ee1b130c0606e4f8bcdf65c1efc2a53)

| bool sf::Shader::loadFromStream | ( | [InputStream](http://docs.google.com/classsf_1_1InputStream.htm) & | *stream*, |
| --- | --- | --- | --- |
|  |  | [Type](http://docs.google.com/classsf_1_1Shader.htm#afaa1aa65e5de37b74d047da9def9f9b3) | *type* |
|  | ) |  |  |

Load either the vertex or fragment shader from a custom stream.

This function loads a single shader, either vertex or fragment, identified by the second argument. The source code must be a valid shader in GLSL language. GLSL is a C-like language dedicated to OpenGL shaders; you'll probably need to read a good documentation for it before writing your own shaders.

Parameters

| stream | Source stream to read from |
| --- | --- |
| type | Type of shader (vertex or fragment) |

ReturnsTrue if loading succeeded, false if it failed See Also[loadFromFile](http://docs.google.com/classsf_1_1Shader.htm#a053a5632848ebaca2fcd8ba29abe9e6e), [loadFromMemory](http://docs.google.com/classsf_1_1Shader.htm#ac92d46bf71dff2d791117e4e472148aa)

| bool sf::Shader::loadFromStream | ( | [InputStream](http://docs.google.com/classsf_1_1InputStream.htm) & | *vertexShaderStream*, |
| --- | --- | --- | --- |
|  |  | [InputStream](http://docs.google.com/classsf_1_1InputStream.htm) & | *fragmentShaderStream* |
|  | ) |  |  |

Load both the vertex and fragment shaders from custom streams.

This function loads both the vertex and the fragment shaders. If one of them fails to load, the shader is left empty (the valid shader is unloaded). The source codes must be valid shaders in GLSL language. GLSL is a C-like language dedicated to OpenGL shaders; you'll probably need to read a good documentation for it before writing your own shaders.

Parameters

| vertexShaderStream | Source stream to read the vertex shader from |
| --- | --- |
| fragmentShaderStream | Source stream to read the fragment shader from |

ReturnsTrue if loading succeeded, false if it failed See Also[loadFromFile](http://docs.google.com/classsf_1_1Shader.htm#a053a5632848ebaca2fcd8ba29abe9e6e), [loadFromMemory](http://docs.google.com/classsf_1_1Shader.htm#ac92d46bf71dff2d791117e4e472148aa)

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | float | *x* |
|  | ) |  |  |

Change a float parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a float (float GLSL type).

Example:

uniform float myparam; // this is the variable in the shader

shader.setParameter("myparam", 5.2f);

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| x | Value to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | float | *x*, |
|  |  | float | *y* |
|  | ) |  |  |

Change a 2-components vector parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 2x1 vector (vec2 GLSL type).

Example:

uniform vec2 myparam; // this is the variable in the shader

shader.setParameter("myparam", 5.2f, 6.0f);

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| x | First component of the value to assign |
| y | Second component of the value to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | float | *x*, |
|  |  | float | *y*, |
|  |  | float | *z* |
|  | ) |  |  |

Change a 3-components vector parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 3x1 vector (vec3 GLSL type).

Example:

uniform vec3 myparam; // this is the variable in the shader

shader.setParameter("myparam", 5.2f, 6.0f, -8.1f);

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| x | First component of the value to assign |
| y | Second component of the value to assign |
| z | Third component of the value to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | float | *x*, |
|  |  | float | *y*, |
|  |  | float | *z*, |
|  |  | float | *w* |
|  | ) |  |  |

Change a 4-components vector parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 4x1 vector (vec4 GLSL type).

Example:

uniform vec4 myparam; // this is the variable in the shader

shader.setParameter("myparam", 5.2f, 6.0f, -8.1f, 0.4f);

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| x | First component of the value to assign |
| y | Second component of the value to assign |
| z | Third component of the value to assign |
| w | Fourth component of the value to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | const [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) & | *vector* |
|  | ) |  |  |

Change a 2-components vector parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 2x1 vector (vec2 GLSL type).

Example:

uniform vec2 myparam; // this is the variable in the shader

shader.setParameter("myparam", [sf::Vector2f](http://docs.google.com/classsf_1_1Vector2.htm)(5.2f, 6.0f));

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| vector | Vector to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | const [Vector3f](http://docs.google.com/classsf_1_1Vector3.htm) & | *vector* |
|  | ) |  |  |

Change a 3-components vector parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 3x1 vector (vec3 GLSL type).

Example:

uniform vec3 myparam; // this is the variable in the shader

shader.setParameter("myparam", [sf::Vector3f](http://docs.google.com/classsf_1_1Vector3.htm)(5.2f, 6.0f, -8.1f));

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| vector | Vector to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | const [Color](http://docs.google.com/classsf_1_1Color.htm) & | *color* |
|  | ) |  |  |

Change a color parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 4x1 vector (vec4 GLSL type).

It is important to note that the components of the color are normalized before being passed to the shader. Therefore, they are converted from range [0 .. 255] to range [0 .. 1]. For example, a sf::Color(255, 125, 0, 255) will be transformed to a vec4(1.0, 0.5, 0.0, 1.0) in the shader.

Example:

uniform vec4 color; // this is the variable in the shader

shader.setParameter("color", [sf::Color](http://docs.google.com/classsf_1_1Color.htm)(255, 128, 0, 255));

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| color | [Color](http://docs.google.com/classsf_1_1Color.htm) to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | const [sf::Transform](http://docs.google.com/classsf_1_1Transform.htm) & | *transform* |
|  | ) |  |  |

Change a matrix parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 4x4 matrix (mat4 GLSL type).

Example:

uniform mat4 matrix; // this is the variable in the shader

[sf::Transform](http://docs.google.com/classsf_1_1Transform.htm) transform;

transform.[translate](http://docs.google.com/classsf_1_1Transform.htm#ab54f6c8070cc05e2afcb3145fbf4395a)(5, 10);

shader.setParameter("matrix", transform);

Parameters

| name | Name of the parameter in the shader |
| --- | --- |
| transform | [Transform](http://docs.google.com/classsf_1_1Transform.htm) to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | const [Texture](http://docs.google.com/classsf_1_1Texture.htm) & | *texture* |
|  | ) |  |  |

Change a texture parameter of the shader.

*name* is the name of the variable to change in the shader. The corresponding parameter in the shader must be a 2D texture (sampler2D GLSL type).

Example:

uniform sampler2D the\_texture; // this is the variable in the shader

[sf::Texture](http://docs.google.com/classsf_1_1Texture.htm) texture;

...

shader.setParameter("the\_texture", texture);

It is important to note that *texture* must remain alive as long as the shader uses it, no copy is made internally.

To use the texture of the object being draw, which cannot be known in advance, you can pass the special value sf::Shader::CurrentTexture:

shader.setParameter("the\_texture", sf::Shader::CurrentTexture).

Parameters

| name | Name of the texture in the shader |
| --- | --- |
| texture | [Texture](http://docs.google.com/classsf_1_1Texture.htm) to assign |

| void sf::Shader::setParameter | ( | const std::string & | *name*, |
| --- | --- | --- | --- |
|  |  | [CurrentTextureType](http://docs.google.com/structsf_1_1Shader_1_1CurrentTextureType.htm) |  |
|  | ) |  |  |

Change a texture parameter of the shader.

This overload maps a shader texture variable to the texture of the object being drawn, which cannot be known in advance. The second argument must be sf::Shader::CurrentTexture. The corresponding parameter in the shader must be a 2D texture (sampler2D GLSL type).

Example:

uniform sampler2D current; // this is the variable in the shader

shader.setParameter("current", sf::Shader::CurrentTexture);

Parameters

| name | Name of the texture in the shader |
| --- | --- |

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

* [Shader.hpp](http://docs.google.com/Shader_8hpp_source.htm)

Copyright � Laurent Gomila  ::  Documentation generated by [doxygen](http://www.doxygen.org/)  ::