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**
* [Texture](http://docs.google.com/classsf_1_1Texture.htm)

[Public Types](#_gjdgxs) | [Public Member Functions](#_30j0zll) | [Static Public Member Functions](#_1fob9te) | [Static Private Member Functions](#_3znysh7) | [Friends](#_2et92p0) | [List of all members](http://docs.google.com/classsf_1_1Texture-members.htm)

sf::Texture Class Reference

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

[Image](http://docs.google.com/classsf_1_1Image.htm) living on the graphics card that can be used for drawing. [More...](http://docs.google.com/classsf_1_1Texture.htm#details)

#include <[Texture.hpp](http://docs.google.com/Texture_8hpp_source.htm)>

Inheritance diagram for sf::Texture:



| Public Types | |
| --- | --- |
| enum | [CoordinateType](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82e) {  [Normalized](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82ea69d6228950882e4d68be4ba4dbe7df73),  [Pixels](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82ea6372f9c3a10203a7a69d8d5da59d82ff)  } |
|  | Types of texture coordinates that can be used for rendering. [More...](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82e) |
|  | |

| Public Member Functions | |
| --- | --- |
|  | [Texture](http://docs.google.com/classsf_1_1Texture.htm#a3e04674853b8533bf981db3173e3a4a7) () |
|  | Default constructor. |
|  | |
|  | [Texture](http://docs.google.com/classsf_1_1Texture.htm#a524855cbf89de3b74be84d385fd229de) (const [Texture](http://docs.google.com/classsf_1_1Texture.htm) &copy) |
|  | Copy constructor. |
|  | |
|  | [~Texture](http://docs.google.com/classsf_1_1Texture.htm#a9c5354ad40eb1c5aeeeb21f57ccd7e6c) () |
|  | Destructor. |
|  | |
| bool | [create](http://docs.google.com/classsf_1_1Texture.htm#a89b4c7d204acf1033c3a1b6e0a3ad0a3) (unsigned int width, unsigned int height) |
|  | Create the texture. |
|  | |
| bool | [loadFromFile](http://docs.google.com/classsf_1_1Texture.htm#a8e1b56eabfe33e2e0e1cb03712c7fcc7) (const std::string &filename, const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) &area=[IntRect](http://docs.google.com/classsf_1_1Rect.htm)()) |
|  | Load the texture from a file on disk. |
|  | |
| bool | [loadFromMemory](http://docs.google.com/classsf_1_1Texture.htm#a2c4adb19dd4cbee0a588eeb85e52a249) (const void \*data, std::size\_t size, const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) &area=[IntRect](http://docs.google.com/classsf_1_1Rect.htm)()) |
|  | Load the texture from a file in memory. |
|  | |
| bool | [loadFromStream](http://docs.google.com/classsf_1_1Texture.htm#a6803a13465a7113a8964d1081841886d) ([sf::InputStream](http://docs.google.com/classsf_1_1InputStream.htm) &stream, const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) &area=[IntRect](http://docs.google.com/classsf_1_1Rect.htm)()) |
|  | Load the texture from a custom stream. |
|  | |
| bool | [loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2) (const [Image](http://docs.google.com/classsf_1_1Image.htm) &image, const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) &area=[IntRect](http://docs.google.com/classsf_1_1Rect.htm)()) |
|  | Load the texture from an image. |
|  | |
| [Vector2u](http://docs.google.com/classsf_1_1Vector2.htm) | [getSize](http://docs.google.com/classsf_1_1Texture.htm#a0f370acd8f41c8b97a6959389c521c2c) () const |
|  | Return the size of the texture. |
|  | |
| [Image](http://docs.google.com/classsf_1_1Image.htm) | [copyToImage](http://docs.google.com/classsf_1_1Texture.htm#aefc19bcd95565dd2348fd4cec0facddc) () const |
|  | Copy the texture pixels to an image. |
|  | |
| void | [update](http://docs.google.com/classsf_1_1Texture.htm#ae4eab5c6781316840b0c50ad08370963) (const Uint8 \*pixels) |
|  | Update the whole texture from an array of pixels. |
|  | |
| void | [update](http://docs.google.com/classsf_1_1Texture.htm#a1352d8e16c2aeb4df586ed65dd2c36b9) (const Uint8 \*pixels, unsigned int width, unsigned int height, unsigned int x, unsigned int y) |
|  | Update a part of the texture from an array of pixels. |
|  | |
| void | [update](http://docs.google.com/classsf_1_1Texture.htm#a037cdf171af0fb392d07626a44a4ea17) (const [Image](http://docs.google.com/classsf_1_1Image.htm) &image) |
|  | Update the texture from an image. |
|  | |
| void | [update](http://docs.google.com/classsf_1_1Texture.htm#a87f916490b757fe900798eedf3abf3ba) (const [Image](http://docs.google.com/classsf_1_1Image.htm) &image, unsigned int x, unsigned int y) |
|  | Update a part of the texture from an image. |
|  | |
| void | [update](http://docs.google.com/classsf_1_1Texture.htm#ad3cceef238f7d5d2108a98dd38c17fc5) (const [Window](http://docs.google.com/classsf_1_1Window.htm) &window) |
|  | Update the texture from the contents of a window. |
|  | |
| void | [update](http://docs.google.com/classsf_1_1Texture.htm#a154f246eb8059b602076009ab1cfd175) (const [Window](http://docs.google.com/classsf_1_1Window.htm) &window, unsigned int x, unsigned int y) |
|  | Update a part of the texture from the contents of a window. |
|  | |
| void | [setSmooth](http://docs.google.com/classsf_1_1Texture.htm#a0c3bd6825b9a99714f10d44179d74324) (bool smooth) |
|  | Enable or disable the smooth filter. |
|  | |
| bool | [isSmooth](http://docs.google.com/classsf_1_1Texture.htm#a1d6643d3c76f2be29dc401dc22749e16) () const |
|  | Tell whether the smooth filter is enabled or not. |
|  | |
| void | [setRepeated](http://docs.google.com/classsf_1_1Texture.htm#aaa87d1eff053b9d4d34a24c784a28658) (bool repeated) |
|  | Enable or disable repeating. |
|  | |
| bool | [isRepeated](http://docs.google.com/classsf_1_1Texture.htm#a007a19b48952b7854120bf423c102150) () const |
|  | Tell whether the texture is repeated or not. |
|  | |
| [Texture](http://docs.google.com/classsf_1_1Texture.htm) & | [operator=](http://docs.google.com/classsf_1_1Texture.htm#a80a089b6b19bb09b83012d5f0e6af9ba) (const [Texture](http://docs.google.com/classsf_1_1Texture.htm) &right) |
|  | Overload of assignment operator. |
|  | |

| Static Public Member Functions | |
| --- | --- |
| static void | [bind](http://docs.google.com/classsf_1_1Texture.htm#ae9a4274e7b95ebf7244d09c7445833b0) (const [Texture](http://docs.google.com/classsf_1_1Texture.htm) \*texture, [CoordinateType](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82e) coordinateType=[Normalized](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82ea69d6228950882e4d68be4ba4dbe7df73)) |
|  | Bind a texture for rendering. |
|  | |
| static unsigned int | [getMaximumSize](http://docs.google.com/classsf_1_1Texture.htm#a0bf905d487b104b758549c2e9e20a3fb) () |
|  | Get the maximum texture size allowed. |
|  | |

| 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. |
|  | |

| Friends | |
| --- | --- |
| class | **RenderTexture** |
|  | |
| class | **RenderTarget** |
|  | |

## Detailed Description

[Image](http://docs.google.com/classsf_1_1Image.htm) living on the graphics card that can be used for drawing.

[sf::Texture](http://docs.google.com/classsf_1_1Texture.htm) stores pixels that can be drawn, with a sprite for example.

A texture lives in the graphics card memory, therefore it is very fast to draw a texture to a render target, or copy a render target to a texture (the graphics card can access both directly).

Being stored in the graphics card memory has some drawbacks. A texture cannot be manipulated as freely as a [sf::Image](http://docs.google.com/classsf_1_1Image.htm), you need to prepare the pixels first and then upload them to the texture in a single operation (see [Texture::update](http://docs.google.com/classsf_1_1Texture.htm#ae4eab5c6781316840b0c50ad08370963)).

[sf::Texture](http://docs.google.com/classsf_1_1Texture.htm) makes it easy to convert from/to [sf::Image](http://docs.google.com/classsf_1_1Image.htm), but keep in mind that these calls require transfers between the graphics card and the central memory, therefore they are slow operations.

A texture can be loaded from an image, but also directly from a file/memory/stream. The necessary shortcuts are defined so that you don't need an image first for the most common cases. However, if you want to perform some modifications on the pixels before creating the final texture, you can load your file to a [sf::Image](http://docs.google.com/classsf_1_1Image.htm), do whatever you need with the pixels, and then call [Texture::loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2).

Since they live in the graphics card memory, the pixels of a texture cannot be accessed without a slow copy first. And they cannot be accessed individually. Therefore, if you need to read the texture's pixels (like for pixel-perfect collisions), it is recommended to store the collision information separately, for example in an array of booleans.

Like [sf::Image](http://docs.google.com/classsf_1_1Image.htm), [sf::Texture](http://docs.google.com/classsf_1_1Texture.htm) can handle a unique internal representation of pixels, which is RGBA 32 bits. This means that a pixel must be composed of 8 bits red, green, blue and alpha channels – just like a [sf::Color](http://docs.google.com/classsf_1_1Color.htm).

Usage example:

// This example shows the most common use of sf::Texture:

// drawing a sprite

// Load a texture from a file

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

if (!texture.[loadFromFile](http://docs.google.com/classsf_1_1Texture.htm#a8e1b56eabfe33e2e0e1cb03712c7fcc7)("texture.png"))

return -1;

// Assign it to a sprite

[sf::Sprite](http://docs.google.com/classsf_1_1Sprite.htm) sprite;

sprite.[setTexture](http://docs.google.com/classsf_1_1Sprite.htm#a3729c88d88ac38c19317c18e87242560)(texture);

// Draw the textured sprite

window.draw(sprite);

// This example shows another common use of sf::Texture:

// streaming real-time data, like video frames

// Create an empty texture

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

if (!texture.[create](http://docs.google.com/classsf_1_1Texture.htm#a89b4c7d204acf1033c3a1b6e0a3ad0a3)(640, 480))

return -1;

// Create a sprite that will display the texture

[sf::Sprite](http://docs.google.com/classsf_1_1Sprite.htm) sprite(texture);

while (...) // the main loop

{

...

// update the texture

sf::Uint8\* pixels = ...; // get a fresh chunk of pixels (the next frame of a movie, for example)

texture.[update](http://docs.google.com/classsf_1_1Texture.htm#ae4eab5c6781316840b0c50ad08370963)(pixels);

// draw it

window.draw(sprite);

...

}

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

[sf::Texture::bind](http://docs.google.com/classsf_1_1Texture.htm#ae9a4274e7b95ebf7244d09c7445833b0)(&texture);

... render OpenGL geometry ...

sf::Texture::bind(NULL);

See Also[sf::Sprite](http://docs.google.com/classsf_1_1Sprite.htm), [sf::Image](http://docs.google.com/classsf_1_1Image.htm), [sf::RenderTexture](http://docs.google.com/classsf_1_1RenderTexture.htm)

Definition at line [47](http://docs.google.com/Texture_8hpp_source.htm#l00047) of file [Texture.hpp](http://docs.google.com/Texture_8hpp_source.htm).

## Member Enumeration Documentation

| enum [sf::Texture::CoordinateType](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82e) |
| --- |

Types of texture coordinates that can be used for rendering.

**Enumerator:**

| *Normalized* | [Texture](http://docs.google.com/classsf_1_1Texture.htm) coordinates in range [0 .. 1]. |
| --- | --- |
| *Pixels* | [Texture](http://docs.google.com/classsf_1_1Texture.htm) coordinates in range [0 .. size]. |

Definition at line [55](http://docs.google.com/Texture_8hpp_source.htm#l00055) of file [Texture.hpp](http://docs.google.com/Texture_8hpp_source.htm).

## Constructor & Destructor Documentation

| sf::Texture::Texture | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Default constructor.

Creates an empty texture.

| sf::Texture::Texture | ( | const [Texture](http://docs.google.com/classsf_1_1Texture.htm) & | *copy* | ) |  |
| --- | --- | --- | --- | --- | --- |

Copy constructor.

Parameters

| copy | instance to copy |
| --- | --- |

| sf::Texture::~Texture | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Destructor.

## Member Function Documentation

| | static void sf::Texture::bind | ( | const [Texture](http://docs.google.com/classsf_1_1Texture.htm) \* | *texture*, | | --- | --- | --- | --- | |  |  | [CoordinateType](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82e) | *coordinateType* = [Normalized](http://docs.google.com/classsf_1_1Texture.htm#aa6fd3bbe3c334b3c4428edfb2765a82ea69d6228950882e4d68be4ba4dbe7df73) | |  | ) |  |  | | static |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Bind a texture 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::Texture](http://docs.google.com/classsf_1_1Texture.htm) with OpenGL code.

[sf::Texture](http://docs.google.com/classsf_1_1Texture.htm) t1, t2;

...

sf::Texture::bind(&t1);

// draw OpenGL stuff that use t1...

[sf::Texture::bind](http://docs.google.com/classsf_1_1Texture.htm#ae9a4274e7b95ebf7244d09c7445833b0)(&t2);

// draw OpenGL stuff that use t2...

[sf::Texture::bind](http://docs.google.com/classsf_1_1Texture.htm#ae9a4274e7b95ebf7244d09c7445833b0)(NULL);

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

The *coordinateType* argument controls how texture coordinates will be interpreted. If Normalized (the default), they must be in range [0 .. 1], which is the default way of handling texture coordinates with OpenGL. If Pixels, they must be given in pixels (range [0 .. size]). This mode is used internally by the graphics classes of SFML, it makes the definition of texture coordinates more intuitive for the high-level API, users don't need to compute normalized values.

Parameters

| texture | Pointer to the texture to bind, can be null to use no texture |
| --- | --- |
| coordinateType | Type of texture coordinates to use |

| [Image](http://docs.google.com/classsf_1_1Image.htm) sf::Texture::copyToImage | ( |  | ) | const |
| --- | --- | --- | --- | --- |

Copy the texture pixels to an image.

This function performs a slow operation that downloads the texture's pixels from the graphics card and copies them to a new image, potentially applying transformations to pixels if necessary (texture may be padded or flipped).

Returns[Image](http://docs.google.com/classsf_1_1Image.htm) containing the texture's pixels See Also[loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2)

| bool sf::Texture::create | ( | unsigned int | *width*, |
| --- | --- | --- | --- |
|  |  | unsigned int | *height* |
|  | ) |  |  |

Create the texture.

If this function fails, the texture is left unchanged.

Parameters

| width | Width of the texture |
| --- | --- |
| height | Height of the texture |

ReturnsTrue if creation was successful

| | static unsigned int sf::Texture::getMaximumSize | ( |  | ) |  | | --- | --- | --- | --- | --- | | static |
| --- | --- | --- | --- | --- | --- | --- |

Get the maximum texture size allowed.

This maximum size is defined by the graphics driver. You can expect a value of 512 pixels for low-end graphics card, and up to 8192 pixels or more for newer hardware.

ReturnsMaximum size allowed for textures, in pixels

| [Vector2u](http://docs.google.com/classsf_1_1Vector2.htm) sf::Texture::getSize | ( |  | ) | const |
| --- | --- | --- | --- | --- |

Return the size of the texture.

ReturnsSize in pixels

| bool sf::Texture::isRepeated | ( |  | ) | const |
| --- | --- | --- | --- | --- |

Tell whether the texture is repeated or not.

ReturnsTrue if repeat mode is enabled, false if it is disabled See Also[setRepeated](http://docs.google.com/classsf_1_1Texture.htm#aaa87d1eff053b9d4d34a24c784a28658)

| bool sf::Texture::isSmooth | ( |  | ) | const |
| --- | --- | --- | --- | --- |

Tell whether the smooth filter is enabled or not.

ReturnsTrue if smoothing is enabled, false if it is disabled See Also[setSmooth](http://docs.google.com/classsf_1_1Texture.htm#a0c3bd6825b9a99714f10d44179d74324)

| bool sf::Texture::loadFromFile | ( | const std::string & | *filename*, |
| --- | --- | --- | --- |
|  |  | const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) & | *area* = [IntRect](http://docs.google.com/classsf_1_1Rect.htm)() |
|  | ) |  |  |

Load the texture from a file on disk.

This function is a shortcut for the following code:

[sf::Image](http://docs.google.com/classsf_1_1Image.htm) image;

image.[loadFromFile](http://docs.google.com/classsf_1_1Image.htm#a9e4f2aa8e36d0cabde5ed5a4ef80290b)(filename);

texture.[loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2)(image, area);

The *area* argument can be used to load only a sub-rectangle of the whole image. If you want the entire image then leave the default value (which is an empty IntRect). If the *area* rectangle crosses the bounds of the image, it is adjusted to fit the image size.

The maximum size for a texture depends on the graphics driver and can be retrieved with the getMaximumSize function.

If this function fails, the texture is left unchanged.

Parameters

| filename | Path of the image file to load |
| --- | --- |
| area | Area of the image to load |

ReturnsTrue if loading was successful See Also[loadFromMemory](http://docs.google.com/classsf_1_1Texture.htm#a2c4adb19dd4cbee0a588eeb85e52a249), [loadFromStream](http://docs.google.com/classsf_1_1Texture.htm#a6803a13465a7113a8964d1081841886d), [loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2)

| bool sf::Texture::loadFromImage | ( | const [Image](http://docs.google.com/classsf_1_1Image.htm) & | *image*, |
| --- | --- | --- | --- |
|  |  | const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) & | *area* = [IntRect](http://docs.google.com/classsf_1_1Rect.htm)() |
|  | ) |  |  |

Load the texture from an image.

The *area* argument can be used to load only a sub-rectangle of the whole image. If you want the entire image then leave the default value (which is an empty IntRect). If the *area* rectangle crosses the bounds of the image, it is adjusted to fit the image size.

The maximum size for a texture depends on the graphics driver and can be retrieved with the getMaximumSize function.

If this function fails, the texture is left unchanged.

Parameters

| image | [Image](http://docs.google.com/classsf_1_1Image.htm) to load into the texture |
| --- | --- |
| area | Area of the image to load |

ReturnsTrue if loading was successful See Also[loadFromFile](http://docs.google.com/classsf_1_1Texture.htm#a8e1b56eabfe33e2e0e1cb03712c7fcc7), [loadFromMemory](http://docs.google.com/classsf_1_1Texture.htm#a2c4adb19dd4cbee0a588eeb85e52a249)

| bool sf::Texture::loadFromMemory | ( | const void \* | *data*, |
| --- | --- | --- | --- |
|  |  | std::size\_t | *size*, |
|  |  | const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) & | *area* = [IntRect](http://docs.google.com/classsf_1_1Rect.htm)() |
|  | ) |  |  |

Load the texture from a file in memory.

This function is a shortcut for the following code:

[sf::Image](http://docs.google.com/classsf_1_1Image.htm) image;

image.[loadFromMemory](http://docs.google.com/classsf_1_1Image.htm#aaa6c7afa5851a51cec6ab438faa7354c)(data, size);

texture.[loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2)(image, area);

The *area* argument can be used to load only a sub-rectangle of the whole image. If you want the entire image then leave the default value (which is an empty IntRect). If the *area* rectangle crosses the bounds of the image, it is adjusted to fit the image size.

The maximum size for a texture depends on the graphics driver and can be retrieved with the getMaximumSize function.

If this function fails, the texture is left unchanged.

Parameters

| data | Pointer to the file data in memory |
| --- | --- |
| size | Size of the data to load, in bytes |
| area | Area of the image to load |

ReturnsTrue if loading was successful See Also[loadFromFile](http://docs.google.com/classsf_1_1Texture.htm#a8e1b56eabfe33e2e0e1cb03712c7fcc7), [loadFromStream](http://docs.google.com/classsf_1_1Texture.htm#a6803a13465a7113a8964d1081841886d), [loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2)

| bool sf::Texture::loadFromStream | ( | [sf::InputStream](http://docs.google.com/classsf_1_1InputStream.htm) & | *stream*, |
| --- | --- | --- | --- |
|  |  | const [IntRect](http://docs.google.com/classsf_1_1Rect.htm) & | *area* = [IntRect](http://docs.google.com/classsf_1_1Rect.htm)() |
|  | ) |  |  |

Load the texture from a custom stream.

This function is a shortcut for the following code:

[sf::Image](http://docs.google.com/classsf_1_1Image.htm) image;

image.[loadFromStream](http://docs.google.com/classsf_1_1Image.htm#a21122ded0e8368bb06ed3b9acfbfb501)(stream);

texture.[loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2)(image, area);

The *area* argument can be used to load only a sub-rectangle of the whole image. If you want the entire image then leave the default value (which is an empty IntRect). If the *area* rectangle crosses the bounds of the image, it is adjusted to fit the image size.

The maximum size for a texture depends on the graphics driver and can be retrieved with the getMaximumSize function.

If this function fails, the texture is left unchanged.

Parameters

| stream | Source stream to read from |
| --- | --- |
| area | Area of the image to load |

ReturnsTrue if loading was successful See Also[loadFromFile](http://docs.google.com/classsf_1_1Texture.htm#a8e1b56eabfe33e2e0e1cb03712c7fcc7), [loadFromMemory](http://docs.google.com/classsf_1_1Texture.htm#a2c4adb19dd4cbee0a588eeb85e52a249), [loadFromImage](http://docs.google.com/classsf_1_1Texture.htm#abec4567ad9856a3596dc74803f26fba2)

| [Texture](http://docs.google.com/classsf_1_1Texture.htm)& sf::Texture::operator= | ( | const [Texture](http://docs.google.com/classsf_1_1Texture.htm) & | *right* | ) |  |
| --- | --- | --- | --- | --- | --- |

Overload of assignment operator.

Parameters

| right | Instance to assign |
| --- | --- |

ReturnsReference to self

| void sf::Texture::setRepeated | ( | bool | *repeated* | ) |  |
| --- | --- | --- | --- | --- | --- |

Enable or disable repeating.

Repeating is involved when using texture coordinates outside the texture rectangle [0, 0, width, height]. In this case, if repeat mode is enabled, the whole texture will be repeated as many times as needed to reach the coordinate (for example, if the X texture coordinate is 3 \* width, the texture will be repeated 3 times). If repeat mode is disabled, the "extra space" will instead be filled with border pixels. Warning: on very old graphics cards, white pixels may appear when the texture is repeated. With such cards, repeat mode can be used reliably only if the texture has power-of-two dimensions (such as 256x128). Repeating is disabled by default.

Parameters

| repeated | True to repeat the texture, false to disable repeating |
| --- | --- |

See Also[isRepeated](http://docs.google.com/classsf_1_1Texture.htm#a007a19b48952b7854120bf423c102150)

| void sf::Texture::setSmooth | ( | bool | *smooth* | ) |  |
| --- | --- | --- | --- | --- | --- |

Enable or disable the smooth filter.

When the filter is activated, the texture appears smoother so that pixels are less noticeable. However if you want the texture to look exactly the same as its source file, you should leave it disabled. The smooth filter is disabled by default.

Parameters

| smooth | True to enable smoothing, false to disable it |
| --- | --- |

See Also[isSmooth](http://docs.google.com/classsf_1_1Texture.htm#a1d6643d3c76f2be29dc401dc22749e16)

| void sf::Texture::update | ( | const Uint8 \* | *pixels* | ) |  |
| --- | --- | --- | --- | --- | --- |

Update the whole texture from an array of pixels.

The *pixel* array is assumed to have the same size as the *area* rectangle, and to contain 32-bits RGBA pixels.

No additional check is performed on the size of the pixel array, passing invalid arguments will lead to an undefined behaviour.

This function does nothing if *pixels* is null or if the texture was not previously created.

Parameters

| pixels | Array of pixels to copy to the texture |
| --- | --- |

| void sf::Texture::update | ( | const Uint8 \* | *pixels*, |
| --- | --- | --- | --- |
|  |  | unsigned int | *width*, |
|  |  | unsigned int | *height*, |
|  |  | unsigned int | *x*, |
|  |  | unsigned int | *y* |
|  | ) |  |  |

Update a part of the texture from an array of pixels.

The size of the *pixel* array must match the *width* and *height* arguments, and it must contain 32-bits RGBA pixels.

No additional check is performed on the size of the pixel array or the bounds of the area to update, passing invalid arguments will lead to an undefined behaviour.

This function does nothing if *pixels* is null or if the texture was not previously created.

Parameters

| pixels | Array of pixels to copy to the texture |
| --- | --- |
| width | Width of the pixel region contained in *pixels* |
| height | Height of the pixel region contained in *pixels* |
| x | X offset in the texture where to copy the source pixels |
| y | Y offset in the texture where to copy the source pixels |

| void sf::Texture::update | ( | const [Image](http://docs.google.com/classsf_1_1Image.htm) & | *image* | ) |  |
| --- | --- | --- | --- | --- | --- |

Update the texture from an image.

Although the source image can be smaller than the texture, this function is usually used for updating the whole texture. The other overload, which has (x, y) additional arguments, is more convenient for updating a sub-area of the texture.

No additional check is performed on the size of the image, passing an image bigger than the texture will lead to an undefined behaviour.

This function does nothing if the texture was not previously created.

Parameters

| image | [Image](http://docs.google.com/classsf_1_1Image.htm) to copy to the texture |
| --- | --- |

| void sf::Texture::update | ( | const [Image](http://docs.google.com/classsf_1_1Image.htm) & | *image*, |
| --- | --- | --- | --- |
|  |  | unsigned int | *x*, |
|  |  | unsigned int | *y* |
|  | ) |  |  |

Update a part of the texture from an image.

No additional check is performed on the size of the image, passing an invalid combination of image size and offset will lead to an undefined behaviour.

This function does nothing if the texture was not previously created.

Parameters

| image | [Image](http://docs.google.com/classsf_1_1Image.htm) to copy to the texture |
| --- | --- |
| x | X offset in the texture where to copy the source image |
| y | Y offset in the texture where to copy the source image |

| void sf::Texture::update | ( | const [Window](http://docs.google.com/classsf_1_1Window.htm) & | *window* | ) |  |
| --- | --- | --- | --- | --- | --- |

Update the texture from the contents of a window.

Although the source window can be smaller than the texture, this function is usually used for updating the whole texture. The other overload, which has (x, y) additional arguments, is more convenient for updating a sub-area of the texture.

No additional check is performed on the size of the window, passing a window bigger than the texture will lead to an undefined behaviour.

This function does nothing if either the texture or the window was not previously created.

Parameters

| window | [Window](http://docs.google.com/classsf_1_1Window.htm) to copy to the texture |
| --- | --- |

| void sf::Texture::update | ( | const [Window](http://docs.google.com/classsf_1_1Window.htm) & | *window*, |
| --- | --- | --- | --- |
|  |  | unsigned int | *x*, |
|  |  | unsigned int | *y* |
|  | ) |  |  |

Update a part of the texture from the contents of a window.

No additional check is performed on the size of the window, passing an invalid combination of window size and offset will lead to an undefined behaviour.

This function does nothing if either the texture or the window was not previously created.

Parameters

| window | [Window](http://docs.google.com/classsf_1_1Window.htm) to copy to the texture |
| --- | --- |
| x | X offset in the texture where to copy the source window |
| y | Y offset in the texture where to copy the source window |

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

* [Texture.hpp](http://docs.google.com/Texture_8hpp_source.htm)

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