sf::Texture Class Reference

Graphics module

Image living on the graphics card that can be used for drawing. More...

`#include <SFML/Graphics/Texture.hpp>`

Inheritance diagram for sf::Texture:

Public Member Functions

	Texture () Default constructor.
	~Texture () Destructor.
	Texture (const Texture &copy) Copy constructor.
Texture &	operator= (const Texture &) Copy assignment operator.
	Texture (Texture &&) noexcept Move constructor.
Texture &	operator= (Texture &&) noexcept Move assignment operator.
	Texture (const std::filesystem::path &filename, bool sRgb=false) Construct the texture from a file on disk.
	Texture (const std::filesystem::path &filename, bool sRgb, const IntRect &area) Construct the texture from a sub-rectangle of a file on disk.
	Texture (const void *data, std::size_t size, bool sRgb=false) Construct the texture from a file in memory.
	Texture (const void *data, std::size_t size, bool sRgb, const IntRect &area)

	Construct the texture from a sub-rectangle of a file in memory.
	Texture (InputStream &stream, bool sRgb=false) Construct the texture from a custom stream.
	Texture (InputStream &stream, bool sRgb, const IntRect &area) Construct the texture from a sub-rectangle of a custom stream.
	Texture (const Image &image, bool sRgb=false) Construct the texture from an image.
	Texture (const Image &image, bool sRgb, const IntRect &area) Construct the texture from a sub-rectangle of an image.
	Texture (Vector2u size, bool sRgb=false) Construct the texture with a given size.
bool	resize (Vector2u size, bool sRgb=false) Resize the texture.
bool	loadFromFile (const std::filesystem::path &filename, bool sRgb=false, const IntRect &area={}) Load the texture from a file on disk.
bool	loadFromMemory (const void *data, std::size_t size, bool sRgb=false, const IntRect &area={}) Load the texture from a file in memory.
bool	loadFromStream (InputStream &stream, bool sRgb=false, const IntRect &area={}) Load the texture from a custom stream.
bool	loadFromImage (const Image &image, bool sRgb=false, const IntRect &area={}) Load the texture from an image.
Vector2u	getSize () const Return the size of the texture.
Image	copyToImage () const Copy the texture pixels to an image.
void	update (const std::uint8_t *pixels) Update the whole texture from an array of pixels.
void	update (const std::uint8_t *pixels, Vector2u size, Vector2u dest) Update a part of the texture from an array of pixels.
void	update (const Texture &texture) Update a part of this texture from another texture.
void	update (const Texture &texture, Vector2u dest) Update a part of this texture from another texture.
void	update (const Image &image) Update the texture from an image.
void	update (const Image &image, Vector2u dest) Update a part of the texture from an image.
void	update (const Window &window)

	Update the texture from the contents of a window.
void	update (const Window &window, Vector2u dest) Update a part of the texture from the contents of a window.
void	setSmooth (bool smooth) Enable or disable the smooth filter.
bool	isSmooth () const Tell whether the smooth filter is enabled or not.
bool	isSrgb () const Tell whether the texture source is converted from sRGB or not.
void	setRepeated (bool repeated) Enable or disable repeating.
bool	isRepeated () const Tell whether the texture is repeated or not.
bool	generateMipmap () Generate a mipmap using the current texture data.
void	swap (Texture &right) noexcept Swap the contents of this texture with those of another.
unsigned int	getNativeHandle () const Get the underlying OpenGL handle of the texture.

Static Public Member Functions

static void	bind (const Texture *texture, CoordinateType coordinateType=CoordinateType::Normalized) Bind a texture for rendering.
static unsigned int	getMaximumSize () Get the maximum texture size allowed.

Friends

class	Text
class	RenderTexture
class	RenderTarget

Detailed Description

Image living on the graphics card that can be used for drawing.

`sf::Texture` 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`, you need to prepare the pixels first and then upload them to the texture in a single operation (see `Texture::update`).

`sf::Texture` makes it easy to convert from/to `sf::Image`, 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`, do whatever you need with the pixels, and then call `Texture(const Image&)`.

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`, `sf::Texture` can handle a unique internal representation of pixels, which is RGBA 32 bits. This means that a pixel must be composed of 8 bit red, green, blue and alpha channels – just like a `sf::Color`.

When providing texture data from an image file or memory, it can either be stored in a linear color space or an sRGB color space. Most digital images account for gamma correction already, so they would need to be "uncorrected" back to linear color space before being processed by the hardware. The hardware can automatically convert it from the sRGB color space to a linear color space when it gets sampled. When the rendered image gets output to the final framebuffer, it gets converted back to sRGB.

This option is only useful in conjunction with an sRGB capable framebuffer. This can be requested during window creation.

Usage example:

```
// This example shows the most common use of sf::Texture:
// drawing a sprite

// Load a texture from a file
const sf::Texture texture("texture.png");

// Assign it to a sprite
sf::Sprite sprite(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 texture({640, 480});

// Create a sprite that will display the texture
sf::Sprite sprite(texture);

while (...) // the main loop
{
    ...

    // update the texture
    std::uint8_t* pixels = ...; // get a fresh chunk of pixels (the next
       frame of a movie, for example)
    texture.update(pixels);

    // draw it
    window.draw(sprite);

    ...
}
```

Like `sf::Shader` that can be used as a raw OpenGL shader, `sf::Texture` can also be used directly as a raw texture for custom OpenGL geometry.

```
sf::Texture::bind(&texture);
... render OpenGL geometry ...
sf::Texture::bind(nullptr);
```

See also

`sf::Sprite`, `sf::Image`, `sf::RenderTexture`

Definition at line 55 of file Texture.hpp.

Constructor & Destructor Documentation

sf::Texture::Texture ( )

Default constructor.

Creates a texture with width 0 and height 0.

See also
`resize`

sf::Texture::~Texture ( )

Destructor.

sf::Texture::Texture (const Texture & copy)

Copy constructor.

Parameters
copy instance to copy

sf::Texture::Texture (Texture &&) noexcept

Move constructor.

sf::Texture::Texture (const std::filesystem::path & filename,
bool sRgb = false) explicit

Construct the texture from a file on disk.

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

Parameters

filename Path of the image file to load
sRgb `true` to enable sRGB conversion, `false` to disable it

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (const std::filesystem::path & filename,
bool sRgb,
const IntRect & area)

Construct the texture from a sub-rectangle of a file on disk.

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.

Parameters

filename Path of the image file to load
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (const void * data,
std::size_t size,
bool sRgb = false)

Construct the texture from a file in memory.

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

Parameters

data Pointer to the file data in memory
size Size of the data to load, in bytes
sRgb `true` to enable sRGB conversion, `false` to disable it

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (const void * data,
std::size_t size,
bool sRgb,
const IntRect & area)

Construct the texture from a sub-rectangle of a file in memory.

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.

Parameters

data Pointer to the file data in memory
size Size of the data to load, in bytes
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (InputStream & stream,
bool sRgb = false) explicit

Construct the texture from a custom stream.

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

Parameters

stream Source stream to read from
sRgb `true` to enable sRGB conversion, `false` to disable it

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (InputStream & stream,
bool sRgb,
const IntRect & area)

Construct the texture from a sub-rectangle of a custom stream.

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.

Parameters

stream Source stream to read from
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (const Image & image,
bool sRgb = false) explicit

Construct the texture from an image.

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

Parameters

image Image to load into the texture
sRgb `true` to enable sRGB conversion, `false` to disable it

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (const Image & image,
bool sRgb,
const IntRect & area)

Construct the texture from a sub-rectangle of an image.

The `area` argument is used to load only a sub-rectangle of the whole image. 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.

Parameters

image Image to load into the texture
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Exceptions

sf::Exception if loading was unsuccessful

See also

`loadFromFile`, `loadFromMemory`, `loadFromStream`, `loadFromImage`

sf::Texture::Texture (Vector2u size,
bool sRgb = false) explicit

Construct the texture with a given size.

Parameters

size Width and height of the texture
sRgb `true` to enable sRGB conversion, `false` to disable it

Exceptions

sf::Exception if construction was unsuccessful

Member Function Documentation

static void
sf::Texture::bind (const Texture * texture,
CoordinateType coordinateType = CoordinateType::Normalized) 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` with OpenGL code.

```
sf::Texture t1, t2;
...
sf::Texture::bind(&t1);
// draw OpenGL stuff that use t1...
sf::Texture::bind(&t2);
// draw OpenGL stuff that use t2...
sf::Texture::bind(nullptr);
// 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 sf::Texture::copyToImage ( ) const nodiscard

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 containing the texture's pixels

See also

`loadFromImage`

bool sf::Texture::generateMipmap ( ) nodiscard

Generate a mipmap using the current texture data.

Mipmaps are pre-computed chains of optimized textures. Each level of texture in a mipmap is generated by halving each of the previous level's dimensions. This is done until the final level has the size of 1x1. The textures generated in this process may make use of more advanced filters which might improve the visual quality of textures when they are applied to objects much smaller than they are. This is known as minification. Because fewer texels (texture elements) have to be sampled from when heavily minified, usage of mipmaps can also improve rendering performance in certain scenarios.

Mipmap generation relies on the necessary OpenGL extension being available. If it is unavailable or generation fails due to another reason, this function will return `false`. Mipmap data is only valid from the time it is generated until the next time the base level image is modified, at which point this function will have to be called again to regenerate it.

Returns

`true` if mipmap generation was successful, `false` if unsuccessful

static unsigned int sf::Texture::getMaximumSize ( ) static nodiscard

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.

Returns

Maximum size allowed for textures, in pixels

unsigned int sf::Texture::getNativeHandle ( ) const nodiscard

Get the underlying OpenGL handle of the texture.

You shouldn't need to use this function, unless you have very specific stuff to implement that SFML doesn't support, or implement a temporary workaround until a bug is fixed.

Returns

OpenGL handle of the texture or 0 if not yet created

Vector2u sf::Texture::getSize ( ) const nodiscard

Return the size of the texture.

Returns
Size in pixels

bool sf::Texture::isRepeated ( ) const nodiscard

Tell whether the texture is repeated or not.

Returns
`true` if repeat mode is enabled, `false` if it is disabled

See also
`setRepeated`

bool sf::Texture::isSmooth ( ) const nodiscard

Tell whether the smooth filter is enabled or not.

Returns
`true` if smoothing is enabled, `false` if it is disabled

See also
`setSmooth`

bool sf::Texture::isSrgb ( ) const nodiscard

Tell whether the texture source is converted from sRGB or not.

Returns
`true` if the texture source is converted from sRGB, `false` if not

See also
`setSrgb`

bool sf::Texture::loadFromFile (const std::filesystem::path & filename,
bool sRgb = false,
const IntRect & area = {}) nodiscard

Load the texture from a file on disk.

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
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Returns

`true` if loading was successful, `false` if it failed

See also

`loadFromMemory`, `loadFromStream`, `loadFromImage`

bool sf::Texture::loadFromImage (const Image & image,
bool sRgb = false,
const IntRect & area = {}) nodiscard

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 to load into the texture
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Returns

`true` if loading was successful, `false` if it failed

See also

`loadFromFile`, `loadFromMemory`

bool sf::Texture::loadFromMemory (const void * data,
std::size_t size,
bool sRgb = false,
const IntRect & area = {}) nodiscard

Load the texture from a file in memory.

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
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Returns

`true` if loading was successful, `false` if it failed

See also

`loadFromFile`, `loadFromStream`, `loadFromImage`

bool sf::Texture::loadFromStream (InputStream & stream,
bool sRgb = false,
const IntRect & area = {}) nodiscard

Load the texture from a custom stream.

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
sRgb `true` to enable sRGB conversion, `false` to disable it
area Area of the image to load

Returns

`true` if loading was successful, `false` if it failed

See also

`loadFromFile`, `loadFromMemory`, `loadFromImage`

Texture & sf::Texture::operator= (const Texture &)

Copy assignment operator.

Texture & sf::Texture::operator= (Texture &&) noexcept

Move assignment operator.

bool sf::Texture::resize (Vector2u size,
bool sRgb = false) nodiscard

Resize the texture.

If this function fails, the texture is left unchanged.

Parameters

size Width and height of the texture
sRgb `true` to enable sRGB conversion, `false` to disable it

Returns

`true` if resizing was successful, `false` if it failed

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`

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`

void sf::Texture::swap (Texture & right) noexcept

Swap the contents of this texture with those of another.

Parameters

right Instance to swap with

void sf::Texture::update (const Image & 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 an additional destination argument, 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 behavior.

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

Parameters

image Image to copy to the texture

void sf::Texture::update (const Image & image,
Vector2u dest)

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 destination will lead to an undefined behavior.

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

Parameters

image Image to copy to the texture
dest Coordinates of the destination position

void sf::Texture::update (const std::uint8_t * 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 behavior.

This function does nothing if `pixels` is `nullptr` or if the texture was not previously created.

Parameters

pixels Array of pixels to copy to the texture

void sf::Texture::update (const std::uint8_t * pixels,
Vector2u size,
Vector2u dest)

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

The size of the pixel array must match the `size` argument, 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 behavior.

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
size Width and height of the pixel region contained in `pixels`
dest Coordinates of the destination position

void sf::Texture::update (const Texture & texture)

Update a part of this texture from another texture.

Although the source texture can be smaller than this texture, this function is usually used for updating the whole texture. The other overload, which has an additional destination argument, is more convenient for updating a sub-area of this texture.

No additional check is performed on the size of the passed texture. Passing a texture bigger than this texture will lead to an undefined behavior.

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

Parameters

texture Source texture to copy to this texture

void sf::Texture::update (const Texture & texture,
Vector2u dest)

Update a part of this texture from another texture.

No additional check is performed on the size of the texture. Passing an invalid combination of texture size and destination will lead to an undefined behavior.

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

Parameters

texture Source texture to copy to this texture
dest Coordinates of the destination position

void sf::Texture::update (const Window & 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 an additional destination argument, 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 behavior.

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

Parameters

window Window to copy to the texture

void sf::Texture::update (const Window & window,
Vector2u dest)

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 destination will lead to an undefined behavior.

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

Parameters

window Window to copy to the texture
dest Coordinates of the destination position

Friends And Related Symbol Documentation

friend class RenderTarget friend

Definition at line 717 of file Texture.hpp.

friend class RenderTexture friend

Definition at line 716 of file Texture.hpp.

friend class Text friend

Definition at line 715 of file Texture.hpp.

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

- Texture.hpp