## Deprecated List

### Class `sf::Event::MouseWheelEvent`
This event is deprecated and potentially inaccurate. Use `MouseWheelScrollEvent` instead.

### Member `sf::LinesStrip`
Use `LineStrip` instead.

### Member `sf::RenderWindow::capture()` const
Use a `sf::Texture` and its `sf::Texture::update(const Window&)` function and `sf::Image` instead.

```cpp
1 sf::Vector2u windowSize = window.getSize();
2 sf::Texture texture;
3 texture.create(windowSize.x, windowSize.y);
4 texture.update(window);
5 sf::Image screenshot = texture.copyToImage();
```

### Member `sf::Shader::setParameter(const std::string &name, const Vector2f &vector)`
Use `setUniform(const std::string&, const Glsl::Vec2&)` instead.

### Member `sf::Shader::setParameter(const std::string &name, float x, float y)`
Use `setUniform(const std::string&, const Glsl::Vec2&)` instead.
### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, const Vector3f &amp;vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, const Glsl::Vec3&amp;) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, const Color &amp;color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, const Glsl::Vec4&amp;) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, const Texture &amp;texture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, const Texture&amp;) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, float x, float y, float z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, const Glsl::Vec3&amp;) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, const Transform &amp;transform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, const Glsl::Mat4&amp;) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, float x, float y, float z, float w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, const Glsl::Vec4&amp;) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, float x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, float) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Shader::setParameter

<table>
<thead>
<tr>
<th>const std::string &amp;name, CurrentTextureType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use setUniform(const std::string&amp;, CurrentTextureType) instead.</td>
</tr>
</tbody>
</table>

### Member sf::Text::getColor () const

| There is now fill and outline colors instead of a single global color. Use getFillColor() or getOutlineColor() instead. |

### Member sf::Text::setColor (const Color &color)

| |
There is now fill and outline colors instead of a single global color. Use setFillColor() or setOutlineColor() instead.

<table>
<thead>
<tr>
<th>Member</th>
<th>sf::TrianglesFan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use</td>
<td>TriangleFan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member</th>
<th>sf::TrianglesStrip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use</td>
<td>TriangleStrip</td>
</tr>
</tbody>
</table>
Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio module</td>
<td>Sounds, streaming (musics or custom sources), recording</td>
</tr>
<tr>
<td>Graphics module</td>
<td>2D graphics module: sprites, text, shapes, ..</td>
</tr>
<tr>
<td>Network module</td>
<td>Socket-based communication, utilities and higher-level protocols</td>
</tr>
<tr>
<td>System module</td>
<td>Base module of SFML, defining various utilities</td>
</tr>
<tr>
<td>Window module</td>
<td>Provides OpenGL-based windows, and abstractions for input handling</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
Classes

Audio module

Sounds, streaming (musics or custom sources), recording, spatialization.
# Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sf::AlResource</strong></td>
<td>Base class for classes that require an OpenAL context. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::InputSoundFile</strong></td>
<td>Provide read access to sound files. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::Listener</strong></td>
<td>The audio listener is the point in the scene from where all the sounds are heard.</td>
</tr>
<tr>
<td><strong>sf::Music</strong></td>
<td>Streamed music played from an audio file. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::OutputSoundFile</strong></td>
<td>Provide write access to sound files. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::Sound</strong></td>
<td>Regular sound that can be played in the audio environment. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::SoundBuffer</strong></td>
<td>Storage for audio samples defining a sound. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::SoundBufferRecorder</strong></td>
<td>Specialized <a href="#">SoundRecorder</a> which stores the captured audio data.</td>
</tr>
<tr>
<td><strong>sf::SoundFileFactory</strong></td>
<td>Manages and instantiates sound file readers and writers. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::SoundFileReader</strong></td>
<td>Abstract base class for sound file decoding. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::SoundFileWriter</strong></td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>sf::SoundRecorder</td>
<td>Abstract base class for capturing sound data.</td>
</tr>
<tr>
<td>sf::SoundSource</td>
<td>Base class defining a sound's properties.</td>
</tr>
<tr>
<td>sf::SoundStream</td>
<td>Abstract base class for streamed audio sources.</td>
</tr>
</tbody>
</table>
Detailed Description

Sounds, streaming (musics or custom sources), recording, spatialization.

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
Base class for classes that require an OpenAL context. More...

#include <AlResource.hpp>

Inheritance diagram for sf::AlResource:
### Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AlResource()</code></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td><code>~AlResource()</code></td>
<td>Destructor. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Base class for classes that require an OpenAL context.

This class is for internal use only, it must be the base of every class that requires an OpenAL context in order to work.

Definition at line 40 of file AlResource.hpp.
Constructor & Destructor Documentation

```cpp
sf::AlResource::AlResource ()
```

Default constructor.

```cpp
sf::AlResource::~AlResource ()
```

Destructor.

The documentation for this class was generated from the following file:
- AlResource.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
Provide read access to sound files. More...

#include <InputSoundFile.hpp>

Inheritance diagram for sf::InputSoundFile:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputSoundFile ()</strong></td>
<td>Default constructor.</td>
</tr>
<tr>
<td><strong>~InputSoundFile ()</strong></td>
<td>Destructor.</td>
</tr>
<tr>
<td><strong>bool openFromFile (const std::string &amp;filename)</strong></td>
<td>Open a sound file from the disk for reading.</td>
</tr>
<tr>
<td>*<em>bool openFromMemory (const void <em>data, std::size_t sizeInBytes)</em></em></td>
<td>Open a sound file in memory for reading.</td>
</tr>
<tr>
<td><strong>bool openFromStream (InputStream &amp;stream)</strong></td>
<td>Open a sound file from a custom stream for reading.</td>
</tr>
<tr>
<td><strong>bool openForWriting (const std::string &amp;filename, unsigned int channelCount, unsigned int sampleRate)</strong></td>
<td>Open the sound file from the disk for writing.</td>
</tr>
<tr>
<td><strong>Uint64 getSampleCount ()</strong> const</td>
<td>Get the total number of audio samples in the file.</td>
</tr>
<tr>
<td><strong>unsigned int getChannelCount ()</strong> const</td>
<td>Get the number of channels used by the sound.</td>
</tr>
<tr>
<td><strong>unsigned int getSampleRate ()</strong> const</td>
<td>Get the sample rate of the sound.</td>
</tr>
<tr>
<td><strong>Time getDuration ()</strong> const</td>
<td>Get the total duration of the sound file.</td>
</tr>
</tbody>
</table>
### void `seek` (Uint64 sampleOffset)
Change the current read position to the given sample offset.

### void `seek` (Time timeOffset)
Change the current read position to the given time offset. More...

### Uint64 `read` (Int16 *samples, Uint64 maxCount)
Read audio samples from the open file. More...
Detailed Description

Provide read access to sound files.

This class decodes audio samples from a sound file.

It is used internally by higher-level classes such as `sf::SoundBuffer` and `sf::Music` if you want to process or analyze audio files without playing them, or if you want to implement your own version of `sf::Music` with more specific features.

Usage example:

```cpp
// Open a sound file
sf::InputSoundFile file;
if (!file.openFromFile("music.ogg"))
    /* error */;

// Print the sound attributes
std::cout << "duration: " << file.getDuration().asSeconds() << std::endl;
std::cout << "channels: " << file.getChannelCount() << std::endl;
std::cout << "sample rate: " << file.getSampleRate() << std::endl;
std::cout << "sample count: " << file.getSampleCount() << std::endl;

// Read and process batches of samples until the end of file is reached
sf::Int16 samples[1024];
sf::Uint64 count;
do {
    count = file.read(samples, 1024);
    // process, analyze, play, convert, or whatever
    // you want to do with the samples...
} while (count > 0);
```

See also

`sf::SoundFileReader`, `sf::OutputSoundFile`

Definition at line 46 of file `InputSoundFile.hpp`. 

## Constructor & Destructor Documentation

### sf::InputSoundFile::InputSoundFile()

Default constructor.

### sf::InputSoundFile::~InputSoundFile()

Destructor.
Member Function Documentation

**unsigned int sf::InputSoundFile::getChannelCount ( ) const**

Get the number of channels used by the sound.

**Returns**

Number of channels (1 = mono, 2 = stereo)

**Time sf::InputSoundFile::getDuration ( ) const**

Get the total duration of the sound file.

This function is provided for convenience, the duration is deduced from other sound file attributes.

**Returns**

Duration of the sound file

**Uint64 sf::InputSoundFile::getSampleCount ( ) const**

Get the total number of audio samples in the file.

**Returns**

Number of samples
unsigned int sf::InputSoundFile::getSampleRate () const

Get the sample rate of the sound.

**Returns**

Sample rate, in samples per second

---

bool sf::InputSoundFile::openForWriting ( const std::string & filename,
                                         unsigned int channelCount,
                                         unsigned int sampleRate )

Open the sound file from the disk for writing.

**Parameters**

- **filename** Path of the sound file to write
- **channelCount** Number of channels in the sound
- **sampleRate** Sample rate of the sound

**Returns**

True if the file was successfully opened

---

bool sf::InputSoundFile::openFromFile ( const std::string & filename )

Open a sound file from the disk for reading.

The supported audio formats are: WAV (PCM only), OGG/Vorbis, FLAC. FLAC and WAV are 8, 16, 24 and 32 bit.

**Parameters**

- **filename** Path of the sound file to load
bool sf::InputSoundFile::openFromMemory (const void * data, std::size_t sizeInBytes)

Open a sound file in memory for reading.

The supported audio formats are: WAV (PCM only), OGG/Vorbis, FLAC. FLAC and WAV are 8, 16, 24 and 32 bit.

Parameters
- **data** Pointer to the file data in memory
- **sizeInBytes** Size of the data to load, in bytes

Returns
True if the file was successfully opened

---

bool sf::InputSoundFile::openFromStream (InputStream & stream)

Open a sound file from a custom stream for reading.

The supported audio formats are: WAV (PCM only), OGG/Vorbis, FLAC. FLAC and WAV are 8, 16, 24 and 32 bit.

Parameters
- **stream** Source stream to read from

Returns
True if the file was successfully opened

```
Uint64 sf::InputSoundFile::read ( Int16 * samples,
                                  Uint64 maxCount )
```

Read audio samples from the open file.

**Parameters**
- **samples** Pointer to the sample array to fill
- **maxCount** Maximum number of samples to read

**Returns**
Number of samples actually read (may be less than `maxCount`)

```
void sf::InputSoundFile::seek ( Uint64 sampleOffset )
```

Change the current read position to the given sample offset.

This function takes a sample offset to provide maximum precision. If you need to jump to a given time, use the other overload.

The sample offset takes the channels into account. Offsets can be calculated like this:

```
sampleRate * channelCount
```

If the given offset exceeds the total number of samples, this function jumps to the end of the sound file.

**Parameters**
- **sampleOffset** Index of the sample to jump to, relative to the beginning
**void sf::InputSoundFile::seek ( Time timeOffset )**

Change the current read position to the given time offset.

Using a time offset is handy but imprecise. If you need an accurate result, consider using the overload which takes a sample offset.

If the given time exceeds to total duration, this function jumps to the end.

**Parameters**

- **timeOffset** Time to jump to, relative to the beginning

The documentation for this class was generated from the following file:
- InputSoundFile.hpp
The audio listener is the point in the scene from where all the sounds are heard.

```cpp
#include <Listener.hpp>
```
### Static Public Member Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>setGlobalVolume(float volume)</td>
<td>Change the global volume of all the sounds and musics.</td>
</tr>
<tr>
<td>static float</td>
<td>getGlobalVolume()</td>
<td>Get the current value of the global volume. More...</td>
</tr>
<tr>
<td>static void</td>
<td>setPosition(float x, float y, float z)</td>
<td>Set the position of the listener in the scene. More...</td>
</tr>
<tr>
<td>static void</td>
<td>setPosition(const Vector3f &amp;position)</td>
<td>Set the position of the listener in the scene. More...</td>
</tr>
<tr>
<td>static Vector3f</td>
<td>getPosition()</td>
<td>Get the current position of the listener in the scene. More...</td>
</tr>
<tr>
<td>static void</td>
<td>setDirection(float x, float y, float z)</td>
<td>Set the forward vector of the listener in the scene. More...</td>
</tr>
<tr>
<td>static void</td>
<td>setDirection(const Vector3f &amp;direction)</td>
<td>Set the forward vector of the listener in the scene. More...</td>
</tr>
<tr>
<td>static Vector3f</td>
<td>getDirection()</td>
<td>Get the current forward vector of the listener in the scene.</td>
</tr>
<tr>
<td>static void</td>
<td>setUpVector(float x, float y, float z)</td>
<td>Set the upward vector of the listener in the scene. More...</td>
</tr>
<tr>
<td>static void</td>
<td>setUpVector(const Vector3f &amp;upVector)</td>
<td>Set the upward vector of the listener in the scene. More...</td>
</tr>
<tr>
<td>static Vector3f</td>
<td>getUpVector()</td>
<td></td>
</tr>
</tbody>
</table>
Get the current upward vector of the listener in the scene.
Detailed Description

The audio listener is the point in the scene from where all the sounds are heard.

The audio listener defines the global properties of the audio environment, and musics are heard.

If `sf::View` is the eyes of the user, then `sf::Listener` is his ears (by the way, they are often linked together – same position, orientation, etc.).

`sf::Listener` is a simple interface, which allows to setup the listener in the 3D audio environment (position, direction and up vector), and to adjust the global volume.

Because the listener is unique in the scene, `sf::Listener` only contains static functions and does not need to be instantiated.

Usage example:

```cpp
// Move the listener to the position (1, 0, -5)
sf::Listener::setPosition(1, 0, -5);

// Make it face the right axis (1, 0, 0)
sf::Listener::setDirection(1, 0, 0);

// Reduce the global volume
sf::Listener::setGlobalVolume(50);
```

Definition at line 42 of file Listener.hpp.
**Member Function Documentation**

```cpp
static Vector3f sf::Listener::getDirection()  
```

Get the current forward vector of the listener in the scene.

**Returns**
- Listener's forward vector (not normalized)

**See also**
- setDirection

```cpp
static float sf::Listener::getGlobalVolume()  
```

Get the current value of the global volume.

**Returns**
- Current global volume, in the range [0, 100]

**See also**
- setGlobalVolume

```cpp
static Vector3f sf::Listener::getPosition()  
```

Get the current position of the listener in the scene.
Returns
Listener's position

See also
setPosition

static Vector3f sf::Listener::getUpVector ( )

Get the current upward vector of the listener in the scene.

Returns
Listener's upward vector (not normalized)

See also
setUpVector

static void sf::Listener::setDirection ( float x,
                                          float y,
                                          float z
                                      )

Set the forward vector of the listener in the scene.

The direction (also called "at vector") is the vector pointing forward.
Together with the up vector, it defines the 3D orientation of the listener.
It doesn't have to be normalized. The default listener's direction is (0, 0, -1).

Parameters
x X coordinate of the listener's direction
y Y coordinate of the listener's direction
z Z coordinate of the listener's direction
static void sf::Listener::setDirection ( const Vector3f & direction )

Set the forward vector of the listener in the scene.

The direction (also called "at vector") is the vector pointing forward from the listener's perspective. Together with the up vector, it defines the 3D orientation of the listener in the scene. The direction vector doesn't have to be normalized. The default listener's direction is (0, 0, -1).

Parameters

direction New listener's direction

See also

getDirection, setUpVector, setPosition

static void sf::Listener::setGlobalVolume ( float volume )

Change the global volume of all the sounds and musics.

The volume is a number between 0 and 100; it is combined with the individual volume of each sound/music. The default value for the volume is 100 (maximum).

Parameters

volume New global volume, in the range [0, 100]

See also

getGlobalVolume
static void sf::Listener::setPosition ( float x, float y, float z )

Set the position of the listener in the scene.

The default listener's position is (0, 0, 0).

Parameters
- x X coordinate of the listener's position
- y Y coordinate of the listener's position
- z Z coordinate of the listener's position

See also
- getPosition, setDirection

static void sf::Listener::setPosition ( const Vector3f & position )

Set the position of the listener in the scene.

The default listener's position is (0, 0, 0).

Parameters
- position New listener's position

See also
- getPosition, setDirection

static void sf::Listener::setUpVector ( float x,
float y, float z
)

Set the upward vector of the listener in the scene.

The up vector is the vector that points upward from the listener's perspective; it defines the 3D orientation of the listener in the scene. The up vector, together with the direction, defines the 3D orientation of the listener in the scene. The up vector doesn't have to be normalized. The default listener's up vector is (0, 1, 0). It is usually not necessary to change it, especially in 2D scenarios.

**Parameters**
- \( x \) X coordinate of the listener's up vector
- \( y \) Y coordinate of the listener's up vector
- \( z \) Z coordinate of the listener's up vector

**See also**
- `getUpVector`, `setDirection`, `setPosition`

---

**static void sf::Listener::setUpVector ( const Vector3f & upVector )**

Set the upward vector of the listener in the scene.

The up vector is the vector that points upward from the listener's perspective; it defines the 3D orientation of the listener in the scene. The up vector, together with the direction, defines the 3D orientation of the listener in the scene. The default listener's up vector is (0, 1, 0). It is usually not necessary to change it, especially in 2D scenarios.

**Parameters**
- `upVector` New listener's up vector

**See also**
getUpVector, setDirection, setPosition

The documentation for this class was generated from the following file:
- Listener.hpp
Streamed music played from an audio file. More...

```cpp
#include <Music.hpp>
```

Inheritance diagram for sf::Music:
### Public Types

```c
enum Status { Stopped, Paused, Playing }
```

Enumeration of the sound source states. More...
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Music ()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>~Music ()</code></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool openFromFile(const std::string &amp;filename)</code></td>
<td>Open a music from an audio file. [More...]</td>
</tr>
<tr>
<td><code>bool openFromMemory(const void *data, std::size_t sizeInBytes)</code></td>
<td>Open a music from an audio file in memory. [More...]</td>
</tr>
<tr>
<td><code>bool openFromStream(InputStream &amp;stream)</code></td>
<td>Open a music from an audio file in a custom stream. [More...]</td>
</tr>
<tr>
<td><code>Time getDuration () const</code></td>
<td>Get the total duration of the music. [More...]</td>
</tr>
<tr>
<td><code>void play ()</code></td>
<td>Start or resume playing the audio stream. [More...]</td>
</tr>
<tr>
<td><code>void pause ()</code></td>
<td>Pause the audio stream. [More...]</td>
</tr>
<tr>
<td><code>void stop ()</code></td>
<td>Stop playing the audio stream. [More...]</td>
</tr>
<tr>
<td><code>unsigned int getChannelCount () const</code></td>
<td>Return the number of channels of the stream. [More...]</td>
</tr>
<tr>
<td><code>unsigned int getSampleRate () const</code></td>
<td></td>
</tr>
</tbody>
</table>
Get the stream sample rate of the stream. More...

<table>
<thead>
<tr>
<th>Status</th>
<th>getStatus () const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the current status of the stream (stopped, paused, playing)</td>
<td></td>
</tr>
</tbody>
</table>

void setPlayingOffset (Time timeOffset)
Change the current playing position of the stream. More...

<table>
<thead>
<tr>
<th>Time</th>
<th>getPlayingOffset () const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the current playing position of the stream. More...</td>
<td></td>
</tr>
</tbody>
</table>

void setLoop (bool loop)
Set whether or not the stream should loop after reaching the end. More...

bool getLoop () const
Tell whether or not the stream is in loop mode. More...

void setPitch (float pitch)
Set the pitch of the sound. More...

void setVolume (float volume)
Set the volume of the sound. More...

void setPosition (float x, float y, float z)
Set the 3D position of the sound in the audio scene. More...

void setPosition (const Vector3f &position)
Set the 3D position of the sound in the audio scene. More...

void setRelativeToListener (bool relative)
Make the sound's position relative to the listener or absolute. More...

void setMinDistance (float distance)
Set the minimum distance of the sound. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setAttenuation (float attenuation)</td>
<td>Set the attenuation factor of the sound.</td>
<td>More</td>
</tr>
<tr>
<td>float getPitch () const</td>
<td>Get the pitch of the sound.</td>
<td>More</td>
</tr>
<tr>
<td>float getVolume () const</td>
<td>Get the volume of the sound.</td>
<td>More</td>
</tr>
<tr>
<td>Vector3f getPosition () const</td>
<td>Get the 3D position of the sound in the audio scene.</td>
<td>More</td>
</tr>
<tr>
<td>bool isRelativeToListener () const</td>
<td>Tell whether the sound's position is relative to the listener or not.</td>
<td></td>
</tr>
<tr>
<td>float getMinDistance () const</td>
<td>Get the minimum distance of the sound.</td>
<td>More</td>
</tr>
<tr>
<td>float getAttenuation () const</td>
<td>Get the attenuation factor of the sound.</td>
<td>More</td>
</tr>
</tbody>
</table>
### Protected Member Functions

<table>
<thead>
<tr>
<th>Function Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual bool <code>onGetData (Chunk &amp;data)</code></td>
<td>Request a new chunk of audio samples from the stream source. More...</td>
</tr>
<tr>
<td>virtual void <code>onSeek (Time timeOffset)</code></td>
<td>Change the current playing position in the stream source. More...</td>
</tr>
<tr>
<td>void <code>initialize (unsigned int channelCount, unsigned int sampleRate)</code></td>
<td>Define the audio stream parameters. More...</td>
</tr>
</tbody>
</table>
## Protected Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsigned int</code></td>
<td>m_source</td>
<td>OpenAL source identifier.</td>
</tr>
</tbody>
</table>

More...
Detailed Description

Streamed music played from an audio file.

Musics are sounds that are streamed rather than completely loaded in memory.

This is especially useful for compressed musics that usually take hours to load, say, when they are uncompressed: by streaming it instead of loading it entirely, you avoid almost no loading delay. This implies that the underlying resource (file, stream or memory buffer) must remain valid for the lifetime of the sf::Music object.

Apart from that, a sf::Music has almost the same features as the sf::SoundBuffer: play/pause/stop it, request its parameters (channels, sample rate), change the way it is played (pitch, volume, 3D position, ...), etc.

As a sound stream, a music is played in its own thread in order not to block the rest of the program. This means that you can leave the music alone after calling play(), it will manage itself very well.

Usage example:

```cpp
// Declare a new music
sf::Music music;

// Open it from an audio file
if (!music.openFromFile("music.ogg"))
{
    // error...
}

// Change some parameters
music.setPosition(0, 1, 10); // change its 3D position
music.setPitch(2); // increase the pitch
music.setVolume(50); // reduce the volume
music.setLoop(true); // make it loop

// Play it
music.play();
```
See also
   sf::Sound, sf::SoundStream

Definition at line 48 of file Music.hpp.
enum sf::SoundSource::Status

Enumeration of the sound source states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopped</td>
<td>Sound is not playing.</td>
</tr>
<tr>
<td>Paused</td>
<td>Sound is paused.</td>
</tr>
<tr>
<td>Playing</td>
<td>Sound is playing.</td>
</tr>
</tbody>
</table>

Definition at line 50 of file SoundSource.hpp.
Constructor & Destructor Documentation

**sf::Music::Music ( )**

Default constructor.

**sf::Music::~Music ( )**

Destructor.
Member Function Documentation

float sf::SoundSource::getAttenuation () const

Get the attenuation factor of the sound.

Returns

Attenuation factor of the sound

See also

setAttenuation, getMinDistance

unsigned int sf::SoundStream::getChannelCount () const

Return the number of channels of the stream.

1 channel means a mono sound, 2 means stereo, etc.

Returns

Number of channels

Time sf::Music::getDuration () const

Get the total duration of the music.

Returns
**bool sf::SoundStream::getLoop ( ) const**

Tell whether or not the stream is in loop mode.

**Returns**
True if the stream is looping, false otherwise

**See also**
setLoop

**float sf::SoundSource::getMinDistance ( ) const**

Get the minimum distance of the sound.

**Returns**
Minimum distance of the sound

**See also**
setMinDistance, getAttenuation

**float sf::SoundSource::getPitch ( ) const**

Get the pitch of the sound.

**Returns**
Pitch of the sound

**See also**
**setPitch**

**Time sf::SoundStream::getPlayingOffset ( ) const**

Get the current playing position of the stream.

**Returns**
- Current playing position, from the beginning of the stream

**See also**
- setPlayingOffset

**Vector3f sf::SoundSource::getPosition ( ) const**

Get the 3D position of the sound in the audio scene.

**Returns**
- Position of the sound

**See also**
- setPosition

**unsigned int sf::SoundStream::getSampleRate ( ) const**

Get the stream sample rate of the stream.

The sample rate is the number of audio samples played per second. The

**Returns**
- Sample rate, in number of samples per second
### Status `sf::SoundStream::getStatus ( ) const`

Get the current status of the stream (stopped, paused, playing)

**Returns**
- Current status

### float `sf::SoundSource::getVolume ( ) const`

Get the volume of the sound.

**Returns**
- Volume of the sound, in the range $[0, 100]$  

**See also**
- `setVolume`

### void `sf::SoundStream::initialize ( unsigned int channelCount, unsigned int sampleRate )`

Define the audio stream parameters.

This function must be called by derived classes as soon as they know the audio stream parameters. Any attempt to manipulate the stream (e.g., `play()`, ...) before calling this function will fail. It can be called multiple times if the settings of the audio stream change, but only when the stream is stopped.

**Parameters**
**channelCount** Number of channels of the stream

**sampleRate** Sample rate, in samples per second

---

bool sf::SoundSource::isRelativeToListener ( ) const

Tell whether the sound's position is relative to the listener or is absolute.

**Returns**

True if the position is relative, false if it's absolute

**See also**

setRelativeToListener

---

virtual bool sf::Music::onGetData ( Chunk & data )

Request a new chunk of audio samples from the stream source.

This function fills the chunk from the next samples to read from the audio.

**Parameters**

- **data** Chunk of data to fill

**Returns**

True to continue playback, false to stop

Implements sf::SoundStream.

---

virtual void sf::Music::onSeek ( Time timeOffset )

Change the current playing position in the stream source.
Parameters

timeOffset New playing position, from the beginning of the music

Implements sf::SoundStream.

---

**bool sf::Music::openFromFile ( const std::string & filename )**

Open a music from an audio file.

This function doesn't start playing the music (call `play()` to do so). See the documentation of `sf::InputSoundFile` for the list of supported formats.

**Warning**

Since the music is not loaded at once but rather streamed continuously, the file must remain accessible until the `sf::Music` object loads a new music or is destroyed.

**Parameters**

- **filename** Path of the music file to open

**Returns**

- True if loading succeeded, false if it failed

**See also**

- `openFromMemory`, `openFromStream`

---

**bool sf::Music::openFromMemory ( const void * data, std::size_t sizeInBytes )**

Open a music from an audio file in memory.
This function doesn't start playing the music (call `play()` to do so). See the documentation of `sf::InputSoundFile` for the list of supported formats.

**Warning**
Since the music is not loaded at once but rather streamed continuously, it is accessible until the `sf::Music` object loads a new music or is destroyed. That is, you can't deallocate the buffer right after calling this function.

**Parameters**
- `data` Pointer to the file data in memory
- `sizeInBytes` Size of the data to load, in bytes

**Returns**
True if loading succeeded, false if it failed

**See also**
`openFromFile`, `openFromStream`

---

### `bool sf::Music::openFromStream (InputStream & stream)`

Open a music from an audio file in a custom stream.

This function doesn't start playing the music (call `play()` to do so). See the documentation of `sf::InputSoundFile` for the list of supported formats.

**Warning**
Since the music is not loaded at once but rather streamed continuously, it is accessible until the `sf::Music` object loads a new music or is destroyed.

**Parameters**
- `stream` Source stream to read from

**Returns**
True if loading succeeded, false if it failed

See also
openFromFile, openFromMemory

void sf::SoundStream::pause ( )

Pause the audio stream.
This function pauses the stream if it was playing, otherwise (stream already
effect.

See also
play, stop

void sf::SoundStream::play ( )

Start or resume playing the audio stream.
This function starts the stream if it was stopped, resumes it if it was |
beginning if it was already playing. This function uses its own thread so |
the program while the stream is played.

See also
pause, stop

void sf::SoundSource::setAttenuation ( float attenuation )

Set the attenuation factor of the sound.
The attenuation is a multiplicative factor which makes the sound more or less loud according to its distance from the listener. An attenuation of 0 will produce a non-attenuated sound, i.e., its volume will always be the same whether it is heard from near or from far. On the other hand, an attenuation value such as 100 will make the sound fade out very quickly as it gets further from the listener. The default value of the attenuation is 1.

**Parameters**

- **attenuation** New attenuation factor of the sound

**See also**

- getAttenuation, setMinDistance

---

**void sf::SoundStream::setLoop ( bool loop )**

Set whether or not the stream should loop after reaching the end.

If set, the stream will restart from beginning after reaching the end and setLoop(false) is called. The default looping state for streams is false.

**Parameters**

- **loop** True to play in loop, false to play once

**See also**

- getLoop

---

**void sf::SoundSource::setMinDistance ( float distance )**

Set the minimum distance of the sound.

The "minimum distance" of a sound is the maximum distance at which it can be heard at its maximum volume. Further than the minimum distance, it will start to fade out according to its attenuation factor.
inside the head of the listener") is an invalid value and is forbidden. The default value of the minimum distance is 1.

**Parameters**

- **distance** New minimum distance of the sound

**See also**

- getMinDistance, setAttenuation

---

### void sf::SoundSource::setPitch ( float pitch )

Set the pitch of the sound.

The pitch represents the perceived fundamental frequency of a sound; it can make a sound acute or grave by changing its pitch. A side effect of changing the pitch is that it changes the speed of the sound as well. The default value for the pitch is 1.

**Parameters**

- **pitch** New pitch to apply to the sound

**See also**

- getPitch

---

### void sf::SoundStream::setPlayingOffset ( Time timeOffset )

Change the current playing position of the stream.

The playing position can be changed when the stream is either paused or playing. Changing the position when the stream is stopped has no effect, since playing the stream would reset its position.

**Parameters**
**timeOffset** New playing position, from the beginning of the stream

See also

getPlayingOffset

```cpp
void sf::SoundSource::setPosition ( float x,
                                  float y,
                                  float z )
```

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position is (0, 0, 0).

**Parameters**

- `x` X coordinate of the position of the sound in the scene
- `y` Y coordinate of the position of the sound in the scene
- `z` Z coordinate of the position of the sound in the scene

See also

getPosition

```cpp
void sf::SoundSource::setPosition ( const Vector3f & position )
```

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position is (0, 0, 0).

**Parameters**
**position** Position of the sound in the scene

**See also**
getPosition

```cpp
text

**void sf::SoundSource::setRelativeToListener ( bool relative )**

Make the sound's position relative to the listener or absolute.

Making a sound relative to the listener will ensure that it will always be relative to the position of the listener. This can be useful for non-spatialized sounds, or sounds produced by the listener, or sounds attached to it. The default value is false (position is absolute).

**Parameters**

relative True to set the position relative, false to set it absolute

**See also**
isRelativeToListener

```cpp
text

**void sf::SoundSource::setVolume ( float volume )**

Set the volume of the sound.

The volume is a value between 0 (mute) and 100 (full volume). The default value for the volume is 100.

**Parameters**

volume Volume of the sound

**See also**
getVolume
Stop playing the audio stream.

This function stops the stream if it was playing or paused, and does not also resets the playing position (unlike `pause()`).

**See also**

- `play`, `pause`
Member Data Documentation

unsigned int sf::SoundSource::m_source

OpenAL source identifier.

Definition at line 274 of file SoundSource.hpp.

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

- Music.hpp
Public Member Functions | List of all members

sf::OutputSoundFile Class Reference
Audio module

Provide write access to sound files. More...

#include <OutputSoundFile.hpp>

Inheritance diagram for sf::OutputSoundFile:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OutputSoundFile()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>~OutputSoundFile()</code></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool openFromFile(const std::string &amp;filename, unsigned int sampleRate)</code></td>
<td>Open the sound file from the disk for writing. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void write(const Int16 *samples, Uint64 count)</code></td>
<td>Write audio samples to the file. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Provide write access to sound files.

This class encodes audio samples to a sound file.

It is used internally by higher-level classes such as \texttt{sf::SoundBuffer}, but create audio files from custom data sources, like generated audio samples.

Usage example:

```cpp
// Create a sound file, ogg/vorbis format, 44100 Hz, stereo
sf::OutputSoundFile file;
if (!file.openFromFile("music.ogg", 44100, 2))
  /* error */;

while (...)
{
    // Read or generate audio samples from your custom source
    std::vector<sf::Int16> samples = ...;

    // Write them to the file
    file.write(samples.data(), samples.size());
}
```

See also

\texttt{sf::SoundFileWriter}, \texttt{sf::InputSoundFile}

Definition at line 44 of file \texttt{OutputSoundFile.hpp}.
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>sf::OutputSoundFile::OutputSoundFile ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default constructor.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf::OutputSoundFile::~OutputSoundFile ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destructor.</td>
</tr>
<tr>
<td>Closes the file if it was still open.</td>
</tr>
</tbody>
</table>
Member Function Documentation

### `bool sf::OutputSoundFile::openFromFile ( const std::string & filename, unsigned int sampleRate, unsigned int channelCount )`

Open the sound file from the disk for writing.

The supported audio formats are: WAV, OGG/Vorbis, FLAC.

**Parameters**

- `filename` Path of the sound file to write
- `sampleRate` Sample rate of the sound
- `channelCount` Number of channels in the sound

**Returns**

True if the file was successfully opened

### `void sf::OutputSoundFile::write ( const Int16 * samples, Uint64 count )`

Write audio samples to the file.

**Parameters**

- `samples` Pointer to the sample array to write
- `count` Number of samples to write
The documentation for this class was generated from the following file:
- OutputSoundFile.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen.
sf::Sound Class Reference

Audio module

Regular sound that can be played in the audio environment. More...

#include <Sound.hpp>

Inheritance diagram for sf::Sound:
Public Types

```c
enum Status { Stopped, Paused, Playing }
Enumeration of the sound source states. More...
```
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sound ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Sound (const SoundBuffer &amp;buffer)</strong></td>
<td>Construct the sound with a buffer. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Sound (const Sound &amp;copy)</strong></td>
<td>Copy constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>~Sound ()</strong></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void play ()</strong></td>
<td>Start or resume playing the sound. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void pause ()</strong></td>
<td>Pause the sound. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void stop ()</strong></td>
<td>Stop playing the sound <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void setBuffer (const SoundBuffer &amp;buffer)</strong></td>
<td>Set the source buffer containing the audio data to play.</td>
</tr>
<tr>
<td><strong>void setLoop (bool loop)</strong></td>
<td>Set whether or not the sound should loop after reaching the end.</td>
</tr>
<tr>
<td><strong>void setPlayingOffset (Time timeOffset)</strong></td>
<td>Change the current playing position of the sound. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const SoundBuffer * getBuffer () const</strong></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Method</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Get the audio buffer attached to the sound.</td>
<td><code>getAudioBuffer()</code> const</td>
</tr>
<tr>
<td>Tell whether or not the sound is in loop mode.</td>
<td><code>getLoop()</code> const</td>
</tr>
<tr>
<td>Get the current playing position of the sound.</td>
<td><code>getPlayingOffset()</code> const</td>
</tr>
<tr>
<td>Get the current status of the sound (stopped, paused, playing).</td>
<td><code>getStatus()</code> const</td>
</tr>
<tr>
<td>Overload of assignment operator.</td>
<td><code>operator=(const Sound &amp;right)</code></td>
</tr>
<tr>
<td>Reset the internal buffer of the sound.</td>
<td><code>resetBuffer()</code></td>
</tr>
<tr>
<td>Set the pitch of the sound.</td>
<td><code>setPitch(float pitch)</code></td>
</tr>
<tr>
<td>Set the volume of the sound.</td>
<td><code>setVolume(float volume)</code></td>
</tr>
<tr>
<td>Set the 3D position of the sound in the audio scene.</td>
<td><code>setPosition(float x, float y, float z)</code></td>
</tr>
<tr>
<td>Set the 3D position of the sound in the audio scene.</td>
<td><code>setPosition(const Vector3f &amp;position)</code></td>
</tr>
<tr>
<td>Make the sound's position relative to the listener or absolute.</td>
<td><code>setRelativeToListener(bool relative)</code></td>
</tr>
<tr>
<td>Set the minimum distance of the sound.</td>
<td><code>setMinDistance(float distance)</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>void setAttenuation (float attenuation)</td>
<td>Set the attenuation factor of the sound.</td>
</tr>
<tr>
<td>float getPitch () const</td>
<td>Get the pitch of the sound.</td>
</tr>
<tr>
<td>float getVolume () const</td>
<td>Get the volume of the sound.</td>
</tr>
<tr>
<td>Vector3f getPosition () const</td>
<td>Get the 3D position of the sound in the audio scene.</td>
</tr>
<tr>
<td>bool isRelativeToListener () const</td>
<td>Tell whether the sound's position is relative to the listener or is absolute.</td>
</tr>
<tr>
<td>float getMinDistance () const</td>
<td>Get the minimum distance of the sound.</td>
</tr>
<tr>
<td>float getAttenuation () const</td>
<td>Get the attenuation factor of the sound.</td>
</tr>
</tbody>
</table>
## Protected Attributes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsigned int</code></td>
<td><code>m_source</code></td>
<td>OpenAL source identifier. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Regular sound that can be played in the audio environment.

**sf::Sound** is the class to use to play sounds.

It provides:

- Control (play, pause, stop)
- Ability to modify output parameters in real-time (pitch, volume, ...)
- 3D spatial features (position, attenuation, ...).

**sf::Sound** is perfect for playing short sounds that can fit in memory and require no latency, like footsteps or gun shots. For longer sounds, like background musics or long speeches based on streaming).

In order to work, a sound must be given a buffer of audio data to play. **sf::SoundBuffer**, and attached to a sound with the **setBuffer()** function. The sound must remain alive as long as the sound uses it. Note that multiple buffers can be used at the same time.

Usage example:

```cpp
sf::SoundBuffer buffer;
buffer.loadFromFile("sound.wav");

sf::Sound sound;
sound.setBuffer(buffer);
sound.play();
```

See also

- **sf::SoundBuffer**, **sf::Music**
Definition at line 45 of file Sound.hpp.
Member Enumeration Documentation

**enum sf::SoundSource::Status**

Enumeration of the sound source states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopped</td>
<td>Sound is not playing.</td>
</tr>
<tr>
<td>Paused</td>
<td>Sound is paused.</td>
</tr>
<tr>
<td>Playing</td>
<td>Sound is playing.</td>
</tr>
</tbody>
</table>

Definition at line 50 of file `SoundSource.hpp`. 
Constructor & Destructor Documentation

sf::Sound::Sound ( )

Default constructor.

sf::Sound::Sound ( const SoundBuffer & buffer )

Construct the sound with a buffer.

Parameters

buffer Sound buffer containing the audio data to play with the sound

sf::Sound::Sound ( const Sound & copy )

Copy constructor.

Parameters

copy Instance to copy

sf::Sound::~Sound ( )

Destructor.
## Member Function Documentation

### float sf::SoundSource::getAttenuation () const

Get the attenuation factor of the sound.

**Returns**

Attenuation factor of the sound

**See also**

`setAttenuation`, `getMinDistance`

### const SoundBuffer* sf::Sound::getBuffer () const

Get the audio buffer attached to the sound.

**Returns**

Sound buffer attached to the sound (can be NULL)

### bool sf::Sound::getLoop () const

Tell whether or not the sound is in loop mode.

**Returns**

True if the sound is looping, false otherwise
### float sf::SoundSource::getMinDistance ( ) const

Get the minimum distance of the sound.

**Returns**

Minimum distance of the sound

**See also**

setMinDistance, getAttenuation

### float sf::SoundSource::getPitch ( ) const

Get the pitch of the sound.

**Returns**

Pitch of the sound

**See also**

setPitch

### Time sf::Sound::getPlayingOffset ( ) const

Get the current playing position of the sound.

**Returns**

Current playing position, from the beginning of the sound
**See also**

`setPlayingOffset`

---

**Vector3f sf::SoundSource::getPosition () const**

Get the 3D position of the sound in the audio scene.

**Returns**

Position of the sound

**See also**

`setPosition`

---

**Status sf::Sound::getStatus () const**

Get the current status of the sound (stopped, paused, playing)

**Returns**

Current status of the sound

---

**float sf::SoundSource::getVolume () const**

Get the volume of the sound.

**Returns**

Volume of the sound, in the range [0, 100]

**See also**

`setVolume`
### bool sf::SoundSource::isRelativeToListener ( ) const

Tell whether the sound's position is relative to the listener or is absolute.

**Returns**

True if the position is relative, false if it's absolute

**See also**

`setRelativeToListener`

### Sound & sf::Sound::operator= ( const Sound & right )

Overload of assignment operator.

**Parameters**

- **right** Instance to assign

**Returns**

Reference to self

### void sf::Sound::pause ( )

Pause the sound.

This function pauses the sound if it was playing, otherwise (sound already effect.

**See also**

`play`, `stop`
**void sf::Sound::play ( )**

Start or resume playing the sound.

This function starts the stream if it was stopped, resumes it if it was paused, or begins if it was already playing. This function uses its own thread so that it doesn't block the rest of the program while the sound is played.

**See also**

`pause, stop`

---

**void sf::Sound::resetBuffer ( )**

Reset the internal buffer of the sound.

This function is for internal use only, you don't have to use it. It is called by the sound uses, when it is destroyed in order to prevent the sound from using a dead buffer.

---

**void sf::SoundSource::setAttenuation ( float attenuation )**

Set the attenuation factor of the sound.

The attenuation is a multiplicative factor which makes the sound more distance from the listener. An attenuation of 0 will produce a non-attenuated sound always be the same whether it is heard from near or from far. On the other hand, an attenuation value such as 100 will make the sound fade out very quickly as it gets further away. The default value of the attenuation is 1.

**Parameters**
**attenuation** New attenuation factor of the sound

See also
getAttenuation, setMinDistance

| void sf::Sound::setBuffer ( const SoundBuffer & buffer ) |
| Set the source buffer containing the audio data to play. |
| It is important to note that the sound buffer is not copied, thus the sf::Sound buffer alive as long as it is attached to the sound. |
| **Parameters** |
| buffer Sound buffer to attach to the sound |
| See also |
| getBuffer |

| void sf::Sound::setLoop ( bool loop ) |
| Set whether or not the sound should loop after reaching the end. |
| If set, the sound will restart from beginning after reaching the end a setLoop(false) is called. The default looping state for sound is false. |
| **Parameters** |
| loop True to play in loop, false to play once |
| See also |
| getLoop |
### void sf::SoundSource::setMinDistance ( float distance )

Set the minimum distance of the sound.

The "minimum distance" of a sound is the maximum distance at which it is heard at its maximum volume. Further than the minimum distance, it will start to fade out according to its attenuation factor. A value of 0 ("inside the head of the listener") is an invalid value and is forbidden. The default value is 1.

**Parameters**
- **distance** New minimum distance of the sound

**See also**
- getMinDistance, setAttenuation

### void sf::SoundSource::setPitch ( float pitch )

Set the pitch of the sound.

The pitch represents the perceived fundamental frequency of a sound; it can be made more acute or grave by changing its pitch. A side effect of changing the pitch is that the playing speed of the sound as well. The default value for the pitch is 1.

**Parameters**
- **pitch** New pitch to apply to the sound

**See also**
- getPitch
void sf::Sound::setPlayingOffset ( Time timeOffset )

Change the current playing position of the sound.

The playing position can be changed when the sound is either paused or playing. Changing the playing position when the sound is stopped has no effect, since playing the sound will reset its position.

Parameters

timeOffset New playing position, from the beginning of the sound

See also

getPlayingOffset

void sf::SoundSource::setPosition ( float x, float y, float z )

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position of a sound is (0, 0, 0).

Parameters

x X coordinate of the position of the sound in the scene
y Y coordinate of the position of the sound in the scene
z Z coordinate of the position of the sound in the scene

See also

ggetPosition
**void sf::SoundSource::setPosition ( const Vector3f & position )**

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position is (0, 0, 0).

**Parameters**

- **position** Position of the sound in the scene

**See also**

- getPosition

**void sf::SoundSource::setRelativeToListener ( bool relative )**

Make the sound's position relative to the listener or absolute.

Making a sound relative to the listener will ensure that it will always be in the position of the listener. This can be useful for non-spatialized sounds, the listener, or sounds attached to it. The default value is false (position is absolute).

**Parameters**

- **relative** True to set the position relative, false to set it absolute

**See also**

- isRelativeToListener

**void sf::SoundSource::setVolume ( float volume )**

Set the volume of the sound.
The volume is a value between 0 (mute) and 100 (full volume). The default value for the volume is 100.

**Parameters**

- **volume** Volume of the sound

**See also**

- `getVolume`

---

**void sf::Sound::stop ( )**

stop playing the sound

This function stops the sound if it was playing or paused, and does not also resets the playing position (unlike `pause()`).

**See also**

- `play`, `pause`
Member Data Documentation

**unsigned int sf::SoundSource::m_source**

OpenAL source identifier.

Definition at line 274 of file SoundSource.hpp.

The documentation for this class was generated from the following file:
- **Sound.hpp**
sf::SoundBuffer Class Reference
Audio module

Storage for audio samples defining a sound. More...

#include <SoundBuffer.hpp>

Inheritance diagram for sf::SoundBuffer:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoundBuffer ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>SoundBuffer (const SoundBuffer &amp;copy)</strong></td>
<td>Copy constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>~SoundBuffer ()</strong></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool loadFromFile (const std::string &amp;filename)</strong></td>
<td>Load the sound buffer from a file. <a href="#">More...</a></td>
</tr>
<tr>
<td>*<em>bool loadFromMemory (const void <em>data, std::size_t sizeInBytes)</em></em></td>
<td>Load the sound buffer from a file in memory. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool loadFromStream (InputStream &amp;stream)</strong></td>
<td>Load the sound buffer from a custom stream. <a href="#">More...</a></td>
</tr>
<tr>
<td>*<em>bool loadFromSamples (const Int16 <em>samples, Uint64 sampleCount, unsigned int channelCount, unsigned int sampleRate)</em></em></td>
<td>Load the sound buffer from an array of audio samples. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool saveToFile (const std::string &amp;filename) const</strong></td>
<td>Save the sound buffer to an audio file. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const Int16 * getSamples () const</strong></td>
<td>Get the array of audio samples stored in the buffer. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Uint64 getSampleCount () const</strong></td>
<td>Get the number of samples stored in the buffer. <a href="#">More...</a></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>unsigned int getsampleRate()</td>
<td>Get the sample rate of the sound.</td>
</tr>
<tr>
<td>unsigned int getChannelCount()</td>
<td>Get the number of channels used by the sound.</td>
</tr>
<tr>
<td>Time getDuration()</td>
<td>Get the total duration of the sound.</td>
</tr>
<tr>
<td>SoundBuffer &amp; operator=()</td>
<td>Overload of assignment operator.</td>
</tr>
<tr>
<td>class</td>
<td>Sound</td>
</tr>
</tbody>
</table>
Detailed Description

Storage for audio samples defining a sound.

A sound buffer holds the data of a sound, which is an array of audio samples.

A sample is a 16 bits signed integer that defines the amplitude of the sound, and then reconstituted by playing these samples at a high rate (for example, the standard rate used for playing CDs). In short, audio samples are like texture pixels; they are similar to a sf::Texture.

A sound buffer can be loaded from a file (see loadFromFile() for the complete list of supported formats), from memory, from a custom stream (see sf::InputStream) or directly from an array of samples. It can also be saved back to a file.

Sound buffers alone are not very useful: they hold the audio data but cannot be played. To do so, you need to use the sf::Sound class, which provides functions to play/pause/stop the sound and to change the way it is outputted (volume, pitch, 3D position, ...). This separation allows better performances: indeed a sf::SoundBuffer is a heavy resource, and any operation on it is slow (often too slow for real-time applications). On the other side, a sf::Sound is a lightweight object that holds the data of a sound buffer and change the way it is played without actually reading the data. It is also possible to bind several sf::Sound instances to the same sf::SoundBuffer.

It is important to note that the sf::Sound instance doesn't copy the buffer. Rather, it holds a reference to it. Thus, a sf::SoundBuffer must not be destructed while it is used by a function that uses a local sf::SoundBuffer instance for loading a sound.

Usage example:

```
// Declare a new sound buffer
sf::SoundBuffer buffer;
```
// Load it from a file
if (!buffer.loadFromFile("sound.wav"))
{
    // error...
}

// Create a sound source and bind it to the buffer
sf::Sound sound1;
sound1.setBuffer(buffer);

// Play the sound
sound1.play();

// Create another sound source bound to the same buffer
sf::Sound sound2;
sound2.setBuffer(buffer);

// Play it with a higher pitch -- the first sound remains unchanged
sound2.setPitch(2);
sound2.play();

See also
    sf::Sound, sf::SoundBufferRecorder

Definition at line 49 of file SoundBuffer.hpp.
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sf::SoundBuffer::SoundBuffer ( )</code></td>
<td>Default constructor.</td>
</tr>
<tr>
<td><code>sf::SoundBuffer::SoundBuffer ( const SoundBuffer &amp; copy )</code></td>
<td>Copy constructor.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td><code>copy</code></td>
<td>Instance to copy</td>
</tr>
<tr>
<td><code>sf::SoundBuffer::~SoundBuffer ( )</code></td>
<td>Destructor.</td>
</tr>
</tbody>
</table>
Member Function Documentation

**unsigned int sf::SoundBuffer::getChannelCount ( ) const**

Get the number of channels used by the sound.

If the sound is mono then the number of channels will be 1, 2 for stereo,

**Returns**

Number of channels

**See also**

gSampleRate, getDuration

---

**Time sf::SoundBuffer::getDuration ( ) const**

Get the total duration of the sound.

**Returns**

Sound duration

**See also**

gSampleRate, getChannelCount

---

**Uint64 sf::SoundBuffer::getSampleCount ( ) const**
Get the number of samples stored in the buffer.

The array of samples can be accessed with the `getSamples()` function.

**Returns**
Number of samples

**See also**
`getSamples`

```cpp
unsigned int sf::SoundBuffer::getSampleRate ( ) const
```

Get the sample rate of the sound.

The sample rate is the number of samples played per second. The higher, the better the quality (for example, 44100 samples/s is CD quality).

**Returns**
Sample rate (number of samples per second)

**See also**
`getChannelCount`, `getDuration`

```cpp
const Int16* sf::SoundBuffer::getSamples ( ) const
```

Get the array of audio samples stored in the buffer.

The format of the returned samples is 16 bits signed integer (`sf::Int16`). this array is given by the `getSampleCount()` function.
**Returns**

Read-only pointer to the array of sound samples

**See also**

getSampleCount

---

```cpp
bool sf::SoundBuffer::loadFromFile ( const std::string & filename )
```

Load the sound buffer from a file.

See the documentation of `sf::InputSoundFile` for the list of supported formats.

**Parameters**

- `filename` Path of the sound file to load

**Returns**

True if loading succeeded, false if it failed

**See also**

loadFromMemory, loadFromStream, loadFromSamples, saveToFile

---

```cpp
bool sf::SoundBuffer::loadFromMemory ( const void * data, std::size_t sizeInBytes )
```

Load the sound buffer from a file in memory.

See the documentation of `sf::InputSoundFile` for the list of supported formats.

**Parameters**

- `data` Pointer to the file data in memory
**sizeInBytes** Size of the data to load, in bytes

**Returns**
True if loading succeeded, false if it failed

**See also**
loadFromFile, loadFromStream, loadFromSamples

---

```cpp
bool sf::SoundBuffer::loadFromSamples ( const Int16 * samples,
                                         Uint64 sampleCount,
                                         unsigned int channelCount,
                                         unsigned int sampleRate )
```

Load the sound buffer from an array of audio samples.

The assumed format of the audio samples is 16 bits signed integer (sf::Int16).

**Parameters**
- **samples** Pointer to the array of samples in memory
- **sampleCount** Number of samples in the array
- **channelCount** Number of channels (1 = mono, 2 = stereo, ...)
- **sampleRate** Sample rate (number of samples to play per second)

**Returns**
True if loading succeeded, false if it failed

**See also**
loadFromFile, loadFromMemory, saveToFile

---

```cpp
bool sf::SoundBuffer::loadFromStream ( InputStream & stream )
```
Load the sound buffer from a custom stream.

See the documentation of sf::InputSoundFile for the list of supported formats.

**Parameters**
- `stream` Source stream to read from

**Returns**
- True if loading succeeded, false if it failed

See also
- `loadFromFile`, `loadFromMemory`, `loadFromSamples`

---

**sf::SoundBuffer** & sf::SoundBuffer::operator=(const sf::SoundBuffer & right)

Overload of assignment operator.

**Parameters**
- `right` Instance to assign

**Returns**
- Reference to self

---

bool sf::SoundBuffer::saveToFile(const std::string & filename)

Save the sound buffer to an audio file.

See the documentation of sf::OutputSoundFile for the list of supported formats.

**Parameters**
- `filename` Path of the sound file to write
Returns

True if saving succeeded, false if it failed

See also

loadFromFile, loadFromMemory, loadFromSamples

The documentation for this class was generated from the following file:
- SoundBuffer.hpp
sf::SoundBufferRecorder Class Reference
Audio module

Specialized SoundRecorder which stores the captured audio data into a sound buffer.

#include <SoundBufferRecorder.hpp>

Inheritance diagram for sf::SoundBufferRecorder:

```
<table>
<thead>
<tr>
<th>sf::AIIResource</th>
<th>sf::SoundRecorder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sf::SoundBufferRecorder</td>
</tr>
</tbody>
</table>
```
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~SoundBufferRecorder()</td>
<td>Destructor</td>
</tr>
<tr>
<td>const SoundBuffer &amp; getBuffer() const</td>
<td>Get the sound buffer containing the captured audio data.</td>
</tr>
<tr>
<td>bool start(unsigned int sampleRate=44100)</td>
<td>Start the capture.</td>
</tr>
<tr>
<td>void stop()</td>
<td>Stop the capture.</td>
</tr>
<tr>
<td>unsigned int getSampleRate() const</td>
<td>Get the sample rate.</td>
</tr>
<tr>
<td>bool setDevice(const std::string &amp;name)</td>
<td>Set the audio capture device.</td>
</tr>
<tr>
<td>const std::string &amp; getDevice() const</td>
<td>Get the name of the current audio capture device.</td>
</tr>
<tr>
<td>void setChannelCount(unsigned int channelCount)</td>
<td>Set the channel count of the audio capture device.</td>
</tr>
<tr>
<td>unsigned int getChannelCount() const</td>
<td>Get the number of channels used by this recorder.</td>
</tr>
</tbody>
</table>
### Static Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static std::vector&lt;std::string&gt; getAvailableDevices()</code></td>
<td>Get a list of the names of all available audio capture devices.</td>
</tr>
<tr>
<td><code>static std::string getDefaultDevice()</code></td>
<td>Get the name of the default audio capture device.</td>
</tr>
<tr>
<td><code>static bool isAvailable()</code></td>
<td>Check if the system supports audio capture.</td>
</tr>
</tbody>
</table>
## Protected Member Functions

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual bool <strong>onStart</strong> ()</td>
<td>Start capturing audio data.</td>
<td>More</td>
</tr>
<tr>
<td>virtual bool <strong>onProcessSamples</strong> (const Int16 *samples, std::size_t sampleCount)</td>
<td>Process a new chunk of recorded samples.</td>
<td>More</td>
</tr>
<tr>
<td>virtual void <strong>onStop</strong> ()</td>
<td>Stop capturing audio data.</td>
<td>More</td>
</tr>
<tr>
<td>void <strong>setProcessingInterval</strong> (Time interval)</td>
<td>Set the processing interval.</td>
<td>More</td>
</tr>
</tbody>
</table>
Detailed Description

Specialized SoundRecorder which stores the captured audio data into a sound buffer.

sf::SoundBufferRecorder allows to access a recorded sound through a sound buffer, played, saved to a file, etc.

It has the same simple interface as its base class (start(), stop()) and adds a function to retrieve the recorded sound buffer (getBuffer()).

As usual, don't forget to call the isAvailable() function before using this class (see more details about this).

Usage example:

```cpp
if (sf::SoundBufferRecorder::isAvailable())
{
    // Record some audio data
    sf::SoundBufferRecorder recorder;
    recorder.start();
    ...
    recorder.stop();

    // Get the buffer containing the captured audio data
    const sf::SoundBuffer& buffer = recorder.getBuffer();

    // Save it to a file (for example...)
    buffer.saveToFile("my_record.ogg");
}
```

See also

sf::SoundRecorder

Definition at line 44 of file SoundBufferRecorder.hpp.
Constructor & Destructor Documentation

sf::SoundBufferRecorder::~SoundBufferRecorder ( )

destructor
Member Function Documentation

```cpp
class sf::SoundRecorder
{
public:
    static std::vector<std::string> getAvailableDevices();
    
    const SoundBuffer& getBuffer() const;
    
    unsigned int getChannelCount() const;
};
```

Get a list of the names of all available audio capture devices.

This function returns a vector of strings, containing the names of all available audio capture devices.

**Returns**

A vector of strings containing the names

---

Get the sound buffer containing the captured audio data.

The sound buffer is valid only after the capture has ended. This function provides read-only access to the internal sound buffer, but it can be copied if you need to make any modifications to it.

**Returns**

Read-only access to the sound buffer

---

Get the number of channels used by this recorder.

Currently only mono and stereo are supported, so the value is either 1 (for mono) or 2 (for stereo).

**Returns**

An unsigned integer representing the number of channels.
Returns
Number of channels

See also
setChannelCount

static std::string sf::SoundRecorder::getDefaultDevice()

Get the name of the default audio capture device.

This function returns the name of the default audio capture device. If none is available, an empty string is returned.

Returns
The name of the default audio capture device

const std::string& sf::SoundRecorder::getDevice()

Get the name of the current audio capture device.

Returns
The name of the current audio capture device

unsigned int sf::SoundRecorder::getSampleRate()

Get the sample rate.

The sample rate defines the number of audio samples captured per second (for example, 44100 samples/sec is CD quality).
Returns
Sample rate, in samples per second

static bool sf::SoundRecorder::isAvailable ( )

Check if the system supports audio capture.
This function should always be called before using the audio capture feature. If it returns false, then any attempt to use sf::SoundRecorder or one of its derived classes will fail.

Returns
True if audio capture is supported, false otherwise

virtual bool sf::SoundBufferRecorder::onProcessSamples ( const Int16* samples, std::size_t sampleCount )

Process a new chunk of recorded samples.

Parameters
samples Pointer to the new chunk of recorded samples
sampleCount Number of samples pointed by samples

Returns
True to continue the capture, or false to stop it

Implements sf::SoundRecorder.

virtual bool sf::SoundBufferRecorder::onStart ( )
Start capturing audio data.

**Returns**
True to start the capture, or false to abort it

Reimplemented from `sf::SoundRecorder`.

```cpp
virtual void sf::SoundBufferRecorder::onStop ()
```

Stop capturing audio data.

Reimplemented from `sf::SoundRecorder`.

```cpp
void sf::SoundRecorder::setChannelCount ( unsigned int channelCount )
```

Set the channel count of the audio capture device.

This method allows you to specify the number of channels used for recording and 16-bit stereo are supported.

**Parameters**
- `channelCount` Number of channels. Currently only mono (1) and stereo (2) are supported.

**See also**
- `getChannelCount`

```cpp
bool sf::SoundRecorder::setDevice ( const std::string & name )
```
Set the audio capture device.

This function sets the audio capture device to the device with the given (i.e.: while recording). If you do so while recording and opening the device fails, it stops the recording.

**Parameters**

name The name of the audio capture device

**Returns**

True, if it was able to set the requested device

**See also**

getAvailableDevices, getDefaultDevice

---

**void sf::SoundRecorder::setProcessingInterval ( Time interval )**

Set the processing interval.

The processing interval controls the period between calls to the onProcessSamples function. You may want to use a small interval if you want to process the recorded data in real-time, for example.

Note: this is only a hint, the actual period may vary. So don't rely on this timing.

The default processing interval is 100 ms.

**Parameters**

interval Processing interval

---

**bool sf::SoundRecorder::start ( unsigned int sampleRate = 44100 )**
Start the capture.

The `sampleRate` parameter defines the number of audio samples captured per second. The higher the rate, the better the quality (for example, 44100 samples/sec is CD quality). This function uses its own thread so that it doesn't block the rest of the program while the capture runs. Please note that only one capture can happen at the same time. You can select which capture device will be used by passing the name to the `setDevice()` method. If none was selected before, the default capture device will be used. You can get a list of the names of all available capture devices by calling `getAvailableDevices()`.

**Parameters**

- `sampleRate` Desired capture rate, in number of samples per second

**Returns**

- True, if start of capture was successful

**See also**

- `stop`, `getAvailableDevices`

---

```cpp
void sf::SoundRecorder::stop ()
```

Stop the capture.

**See also**

- `start`

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

- `SoundBufferRecorder.hpp`
sf::SoundFileFactory Class Reference

Audio module

Manages and instantiates sound file readers and writers. More...

#include <SoundFileFactory.hpp>
Static Public Member Functions

<table>
<thead>
<tr>
<th>Template</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>template&lt;typename T&gt; static void registerReader ()</td>
<td>Register a new reader. More...</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename T&gt; static void unregisterReader ()</td>
<td>Unregister a reader. More...</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename T&gt; static void registerWriter ()</td>
<td>Register a new writer. More...</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename T&gt; static void unregisterWriter ()</td>
<td>Unregister a writer. More...</td>
<td></td>
</tr>
</tbody>
</table>

| static SoundFileReader * createReaderFromFilename (const std::string &filename) | Instantiate the right reader for the given file on disk. |
| static SoundFileReader * createReaderFromMemory (const void *data, std::size_t sizeInBytes) | Instantiate the right codec for the given file in memory. |
| static SoundFileReader * createReaderFromStream (InputStream &stream) | Instantiate the right codec for the given file in stream. |
| static SoundFileWriter * createWriterFromFilename (const std::string &filename) | Instantiate the right writer for the given file on disk. |
Detailed Description

Manages and instantiates sound file readers and writers.

This class is where all the sound file readers and writers are registered.

You should normally only need to use its registration and unregistration function and manipulation are wrapped into the higher-level classes sf::InputSoundFile.

To register a new reader (writer) use the sf::SoundFileFactory::registerReader function. You don't have to call the unregisterReader (unregisterWriter) function, unregister a format before your application ends (typically, when a plugin is unloaded).

Usage example:

```cpp
sf::SoundFileFactory::registerReader<MySoundFileReader>();
sf::SoundFileFactory::registerWriter<MySoundFileWriter>();
```

See also

sf::InputSoundFile, sf::OutputSoundFile, sf::SoundFileReader, sf::SoundFileWriter

Definition at line 46 of file SoundFileFactory.hpp.
Member Function Documentation

```cpp
static SoundFileReader*
sf::SoundFileFactory::createReaderFromFilename(const std::string& filename) { }
```

Instantiate the right reader for the given file on disk.

It's up to the caller to release the returned reader.

**Parameters**

- `filename` Path of the sound file

**Returns**

A new sound file reader that can read the given file, or null if no reader can handle it.

**See also**

- `createReaderFromMemory`, `createReaderFromStream`

```cpp
static SoundFileReader*
sf::SoundFileFactory::createReaderFromMemory() { }
```

Instantiate the right codec for the given file in memory.

It's up to the caller to release the returned reader.

**Parameters**
data Pointer to the file data in memory
sizeInBytes Total size of the file data, in bytes

Returns
A new sound file codec that can read the given file, or null if no codec

See also
createReaderFromFilename, createReaderFromStream

---

```
static SoundFileReader*
sf::SoundFileFactory::createReaderFromStream
```

Instantiate the right codec for the given file in stream.

It's up to the caller to release the returned reader

Parameters
stream Source stream to read from

Returns
A new sound file codec that can read the given file, or null if no codec

See also
createReaderFromFilename, createReaderFromMemory

---

```
static SoundFileWriter*
sf::SoundFileFactory::createWriterFromFilename
```

Instantiate the right writer for the given file on disk.

It's up to the caller to release the returned writer
### Parameters

**filename** Path of the sound file

### Returns

A new sound file writer that can write given file, or null if no writer can handle it.

```cpp
template<typename T >
static void sf::SoundFileFactory::registerReader ( )
```

Register a new reader.

**See also**

unregisterReader

```cpp
template<typename T >
static void sf::SoundFileFactory::registerWriter ( )
```

Register a new writer.

**See also**

unregisterWriter

```cpp
template<typename T >
static void sf::SoundFileFactory::unregisterReader ( )
```

Unregister a reader.

**See also**

registerReader
template<typename T >

static void sf::SoundFileFactory::unregisterWriter ( )

Unregister a writer.

See also
  registerWriter

The documentation for this class was generated from the following file:
  SoundFileFactory.hpp
sf::SoundFileReader Class Reference

Abstract base class for sound file decoding. More...

#include <SoundFileReader.hpp>
## Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure holding the audio properties of a sound file.</td>
</tr>
</tbody>
</table>
## Public Member Functions

<table>
<thead>
<tr>
<th>virtual</th>
<th>~SoundFileReader ()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Virtual destructor. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual bool</th>
<th>open (InputStream &amp;stream, Info &amp;info)=0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Open a sound file for reading. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual void</th>
<th>seek (Uint64 sampleOffset)=0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Change the current read position to the given sample offset.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual Uint64</th>
<th>read (Int16 *samples, Uint64 maxCount)=0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read audio samples from the open file. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Abstract base class for sound file decoding.

This class allows users to read audio file formats not natively supported by SFML, thus expanding the set of supported readable audio formats.

A valid sound file reader must override the open, seek and write functions, as well as providing a static check function; the latter is used by SFML to find a suitable writer for a given input file.

To register a new reader, use the `sf::SoundFileFactory::registerReader` template function.

Usage example:

```cpp
class MySoundFileReader : public sf::SoundFileReader
{
public:
    static bool check(sf::InputStream& stream)
    {
        // typically, read the first few header bytes and check fields that identify the format
        // return true if the reader can handle the format
    }

    virtual bool open(sf::InputStream& stream, Info& info)
    {
        // read the sound file header and fill the sound attributes
        // (channel count, sample count and sample rate)
        // return true on success
    }

    virtual void seek(sf::Uint64 sampleOffset)
    {
        // advance to the sampleOffset-th sample from the beginning of the sound file
    }

    virtual sf::Uint64 read(sf::Int16* samples, sf::Uint64 maxCount)
    {
        // read up to 'maxCount' samples into the 'samples' array,
        // convert them (for example from normalized float) if they are not stored as 16-bits signed integers in the file
        // return the actual number of samples read
    }
};
```
See also

sf::InputSoundFile, sf::SoundFileFactory, sf::SoundFileWriter

Definition at line 43 of file SoundFileReader.hpp.
### virtual sf::SoundFileReader::~SoundFileReader ()

Virtual destructor.

Definition at line 62 of file SoundFileReader.hpp.
Member Function Documentation

virtual bool sf::SoundFileReader::open ( InputStream & stream, Info & info )

Open a sound file for reading.

The provided stream reference is valid as long as the SoundFileReader it during the whole lifetime of the reader.

Parameters

- **stream**: Source stream to read from
- **info**: Structure to fill with the properties of the loaded sound

Returns

True if the file was successfully opened

virtual Uint64 sf::SoundFileReader::read ( Int16 * samples, Uint64 maxCount )

Read audio samples from the open file.

Parameters

- **samples**: Pointer to the sample array to fill
- **maxCount**: Maximum number of samples to read
Returns
Number of samples actually read (may be less than $maxCount$)

**virtual void sf::SoundFileReader::seek (Uint64 sampleOffset)**

Change the current read position to the given sample offset.

The sample offset takes the channels into account. Offsets can be calculated like this:

$$sampleRate \times channelCount$$

If the given offset exceeds the total number of samples, this function must jump to the end of the file.

**Parameters**

- **sampleOffset**  Index of the sample to jump to, relative to the beginning of the file.

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

- **SoundFileReader.hpp**
sf::SoundFileReader::Info Struct Reference

Structure holding the audio properties of a sound file. More...

#include <SoundFileReader.hpp>
# Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint64</td>
<td>sampleCount</td>
<td>Total number of samples in the file. <a href="#">More...</a></td>
</tr>
<tr>
<td>unsigned int</td>
<td>channelCount</td>
<td>Number of channels of the sound. <a href="#">More...</a></td>
</tr>
<tr>
<td>unsigned int</td>
<td>sampleRate</td>
<td>Samples rate of the sound, in samples per second. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Structure holding the audio properties of a sound file.

Definition at line 51 of file SoundFileReader.hpp.
Member Data Documentation

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int sf::SoundFileReader::Info::channelCount</td>
<td>Number of channels of the sound.</td>
</tr>
<tr>
<td></td>
<td>Definition at line 54 of file SoundFileReader.hpp.</td>
</tr>
<tr>
<td></td>
<td><strong>Uint64 sf::SoundFileReader::Info::sampleCount</strong></td>
</tr>
<tr>
<td></td>
<td>Total number of samples in the file.</td>
</tr>
<tr>
<td></td>
<td>Definition at line 53 of file SoundFileReader.hpp.</td>
</tr>
<tr>
<td></td>
<td><strong>unsigned int sf::SoundFileReader::Info::sampleRate</strong></td>
</tr>
<tr>
<td></td>
<td>Samples rate of the sound, in samples per second.</td>
</tr>
<tr>
<td></td>
<td>Definition at line 55 of file SoundFileReader.hpp.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:
- SoundFileReader.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
sf::SoundFileWriter Class Reference

Abstract base class for sound file encoding. More...

#include <SoundFileWriter.hpp>
<table>
<thead>
<tr>
<th>virtual</th>
<th>~SoundFileWriter ()</th>
<th>Virtual destructor. More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual bool</td>
<td>open (const std::string &amp;filename, unsigned int sampleRate, unsigned int channelCount)=0</td>
<td>Open a sound file for writing. More...</td>
</tr>
<tr>
<td>virtual void</td>
<td>write (const Int16 *samples, Uint64 count)=0</td>
<td>Write audio samples to the open file. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Abstract base class for sound file encoding.

This class allows users to write audio file formats not natively supported by SFML, expanding the set of supported writable audio formats.

A valid sound file writer must override the open and write functions, as well as a static check function; the latter is used by SFML to find a suitable writer for a given filename.

To register a new writer, use the `sf::SoundFileFactory::registerWriter` template function.

Usage example:

```cpp
class MySoundFileWriter : public sf::SoundFileWriter
{
public:

    static bool check(const std::string& filename)
    {
        // typically, check the extension
        // return true if the writer can handle the format
    }

    virtual bool open(const std::string& filename, unsigned int sampleRate)
    {
        // open the file 'filename' for writing,
        // write the given sample rate and channel count to the file header
        // return true on success
    }

    virtual void write(const sf::Int16* samples, sf::Uint64 count)
    {
        // write 'count' samples stored at address 'samples',
        // convert them (for example to normalized float) if the format requires
    }
};

sf::SoundFileFactory::registerWriter<MySoundFileWriter>();
```
See also

sf::OutputSoundFile, sf::SoundFileFactory, sf::SoundFileReader

Definition at line 41 of file SoundFileWriter.hpp.
virtual sf::SoundFileWriter::~SoundFileWriter ( )

Virtual destructor.

Definition at line 49 of file SoundFileWriter.hpp.
Member Function Documentation

**virtual bool sf::SoundFileWriter::open ( const std::string & filename, unsigned int sampleRate, unsigned int channelCount )**

Open a sound file for writing.

**Parameters**

- **filename** Path of the file to open
- **sampleRate** Sample rate of the sound
- **channelCount** Number of channels of the sound

**Returns**

True if the file was successfully opened

**virtual void sf::SoundFileWriter::write ( const Int16 * samples, Uint64 count )**

Write audio samples to the open file.

**Parameters**

- **samples** Pointer to the sample array to write
- **count** Number of samples to write
The documentation for this class was generated from the following file:
- SoundFileWriter.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::SoundRecorder Class Reference

Audio module

Abstract base class for capturing sound data. More...

#include <SoundRecorder.hpp>

Inheritance diagram for sf::SoundRecorder:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual</td>
<td>~SoundRecorder</td>
<td>()</td>
<td>destructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>start</td>
<td>(unsigned int sampleRate=44100)</td>
<td>Start the capture.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>stop</td>
<td>()</td>
<td>Stop the capture.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>getSampleRate</td>
<td>() const</td>
<td>Get the sample rate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>setDevice</td>
<td>(const std::string &amp;name)</td>
<td>Set the audio capture device.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const std::string &amp;</td>
<td>getDevice</td>
<td>() const</td>
<td>Get the name of the current audio capture device.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>setChannelCount</td>
<td>(unsigned int channelCount)</td>
<td>Set the channel count of the audio capture device.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>getChannelCount</td>
<td>() const</td>
<td>Get the number of channels used by this recorder.</td>
</tr>
</tbody>
</table>
**Static Public Member Functions**

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static std::vector&lt; std::string &gt;</td>
<td><code>getAvailableDevices ()</code></td>
<td>Get a list of the names of all available audio capture devices.</td>
</tr>
<tr>
<td>static std::string</td>
<td><code>getDefaultDevice ()</code></td>
<td>Get the name of the default audio capture device.</td>
</tr>
<tr>
<td>static bool</td>
<td><code>isAvailable ()</code></td>
<td>Check if the system supports audio capture.</td>
</tr>
</tbody>
</table>
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SoundRecorder()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>void <code>setProcessingInterval(Time interval)</code></td>
<td>Set the processing interval. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual bool <code>onStart()</code></td>
<td>Start capturing audio data. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual bool <code>onProcessSamples(const Int16 *samples, std::size_t sampleCount)=0</code></td>
<td>Process a new chunk of recorded samples. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual void <code>onStop()</code></td>
<td>Stop capturing audio data. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Abstract base class for capturing sound data.

`sf::SoundBuffer` provides a simple interface to access the audio recording capabilities of the computer (the microphone).

As an abstract base class, it only cares about capturing sound samples; useful with them is left to the derived class. Note that SFML provides a built-in specialization for saving the captured data to a sound buffer (see `sf::SoundBufferRecorder`).

A derived class has only one virtual function to override:

- `onProcessSamples` provides the new chunks of audio samples while the capture happens.

Moreover, two additional virtual functions can be overridden as well if necessary:

- `onStart` is called before the capture happens, to perform custom initializations.
- `onStop` is called after the capture ends, to perform custom cleanup.

A derived class can also control the frequency of the `onProcessSamples` calls with the `setProcessingInterval` protected function. The default interval is chosen to consume too much CPU, but it can be changed to a smaller value if you need to process the recorded data in real time, for example.

The audio capture feature may not be supported or activated on every platform; check its availability with the `isAvailable()` function. If it returns false, the usage of the recorder will fail.

If you have multiple sound input devices connected to your computer (for example: microphone, external soundcard, webcam mic, ...) you can get a list of all available devices to
function. You can then select a device by calling `setDevice()` with the appropriate device. Otherwise the default capturing device will be used.

By default the recording is in 16-bit mono. Using the `setChannelCount` method you can change the number of channels used by the audio capture device to record. Note that you have to decide whether to record in mono or stereo before starting the recording.

It is important to note that the audio capture happens in a separate thread of the program. In particular, the `onProcessSamples` virtual function (but not `onStart` and `onStop`) will be called from this separate thread. It is important to keep this in mind, because synchronization issues may arise if you share data between threads. Another thing to bear in mind is that you must call `stop()` in the destructor of your derived class, so that the recording thread finishes before your object is destroyed.

Usage example:

```cpp
class CustomRecorder : public sf::SoundRecorder
{
    ~CustomRecorder()
    {
        // Make sure to stop the recording thread
        stop();
    }

    virtual bool onStart() // optional
    {
        // Initialize whatever has to be done before the capture starts
        ...

        // Return true to start playing
        return true;
    }

    virtual bool onProcessSamples(const Int16* samples, std::size_t sampleCount)
    {
        // Do something with the new chunk of samples (store them, send them, ...

        // Return true to continue playing
        return true;
    }

    virtual void onStop() // optional
    {
        // Clean up whatever has to be done after the capture ends
    }
};
```
...}
}

// Usage
if (CustomRecorder::isAvailable())
{
    CustomRecorder recorder;
    if (!recorder.start())
        return -1;

    ...
    recorder.stop();
}

See also
    sf::SoundBufferRecorder

Definition at line 45 of file SoundRecorder.hpp.
## Constructor & Destructor Documentation

### virtual sf::SoundRecorder::~SoundRecorder ( )

destructor

### sf::SoundRecorder::SoundRecorder ( )

Default constructor.

This constructor is only meant to be called by derived classes.
Member Function Documentation

static std::vector<std::string> sf::SoundRecorder::getAvailableDevices

Get a list of the names of all available audio capture devices.

This function returns a vector of strings, containing the names of all available audio capture devices.

Returns
A vector of strings containing the names

unsigned int sf::SoundRecorder::getChannelCount () const

Get the number of channels used by this recorder.

Currently only mono and stereo are supported, so the value is either 1 (for mono) or 2 (for stereo).

Returns
Number of channels

See also
setChannelCount

static std::string sf::SoundRecorder::getDefaultDevice ()

Get the name of the default audio capture device.
This function returns the name of the default audio capture device. If none is available, an empty string is returned.

**Returns**

The name of the default audio capture device

```cpp
const std::string& sf::SoundRecorder::getDevice() const
```

Get the name of the current audio capture device.

**Returns**

The name of the current audio capture device

```cpp
unsigned int sf::SoundRecorder::getSampleRate() const
```

Get the sample rate.

The sample rate defines the number of audio samples captured per second (for example, 44100 samples/sec is CD quality).

**Returns**

Sample rate, in samples per second

```cpp
static bool sf::SoundRecorder::isAvailable()
```

Check if the system supports audio capture.

This function should always be called before using the audio capture feature; otherwise, an attempt to use `sf::SoundRecorder` or one of its derived classes will fail.
virtual bool sf::SoundRecorder::onProcessSamples (const Int16 * samples, std::size_t sampleCount)

Process a new chunk of recorded samples.

This virtual function is called every time a new chunk of recorded data is available. The derived class can then do whatever it wants with it (storing it, playing it, sending it over the network, etc.).

Parameters

- **samples** Pointer to the new chunk of recorded samples
- **sampleCount** Number of samples pointed by `samples`

Returns

True to continue the capture, or false to stop it

Implemented in `sf::SoundBufferRecorder`.

virtual bool sf::SoundRecorder::onStart ()

Start capturing audio data.

This virtual function may be overridden by a derived class if something has to be done every time a capture starts. If not, this function can be ignored; the default implementation does nothing.

Returns

True to start the capture, or false to abort it
virtual void sf::SoundRecorder::onStop ()

Stop capturing audio data.

This virtual function may be overridden by a derived class if something capture ends. If not, this function can be ignored; the default implementa

Reimplemented in sf::SoundBufferRecorder.

void sf::SoundRecorder::setChannelCount ( unsigned int channelCount )

Set the channel count of the audio capture device.

This method allows you to specify the number of channels used for reco and 16-bit stereo are supported.

Parameters

channelCount Number of channels. Currently only mono (1) and st

See also

getchannelCount

bool sf::SoundRecorder::setDevice ( const std::string & name )

Set the audio capture device.

This function sets the audio capture device to the device with the given
(i.e: while recording). If you do so while recording and opening the device fails, it stops the recording.

**Parameters**

- **name** The name of the audio capture device

**Returns**

True, if it was able to set the requested device

**See also**

getAvailableDevices, getDefaultDevice

### void sf::SoundRecorder::setProcessingInterval ( Time interval )

Set the processing interval.

The processing interval controls the period between calls to the onProcessSamples function. You may want to use a small interval if you want to process the recorded data in real-time, for example.

Note: this is only a hint, the actual period may vary. So don't rely on this timing.

The default processing interval is 100 ms.

**Parameters**

- **interval** Processing interval

### bool sf::SoundRecorder::start ( unsigned int sampleRate = 44100 )

Start the capture.

The `sampleRate` parameter defines the number of audio samples captured.
better the quality (for example, 44100 samples/sec is CD quality). This
that it doesn't block the rest of the program while the capture runs. Please
happen at the same time. You can select which capture device will be use
setDevice() method. If none was selected before, the default capture de
list of the names of all available capture devices by calling getAvailableD

Parameters

    sampleRate Desired capture rate, in number of samples per second

Returns

    True, if start of capture was successful

See also

    stop, getAvailableDevices

void sf::SoundRecorder::stop ( )

Stop the capture.

See also

    start

The documentation for this class was generated from the following file:
    SoundRecorder.hpp
sf::SoundSource Class Reference

Audio module

Base class defining a sound's properties. More...

#include <SoundSource.hpp>

Inheritance diagram for sf::SoundSource:
Public Types

```
enum Status { Stopped, Paused, Playing }
Enumeration of the sound source states. More...
```
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoundSource (const SoundSource &amp;copy)</strong></td>
<td>Copy constructor. More...</td>
</tr>
<tr>
<td>virtual ~SoundSource ()</td>
<td>Destructor. More...</td>
</tr>
<tr>
<td>void setPitch (float pitch)</td>
<td>Set the pitch of the sound. More...</td>
</tr>
<tr>
<td>void setVolume (float volume)</td>
<td>Set the volume of the sound. More...</td>
</tr>
<tr>
<td>void setPosition (float x, float y, float z)</td>
<td>Set the 3D position of the sound in the audio scene. More...</td>
</tr>
<tr>
<td>void setPosition (const Vector3f &amp;position)</td>
<td>Set the 3D position of the sound in the audio scene. More...</td>
</tr>
<tr>
<td>void setRelativeToListener (bool relative)</td>
<td>Make the sound's position relative to the listener or abs...</td>
</tr>
<tr>
<td>void setMinDistance (float distance)</td>
<td>Set the minimum distance of the sound. More...</td>
</tr>
<tr>
<td>void setAttenuation (float attenuation)</td>
<td>Set the attenuation factor of the sound. More...</td>
</tr>
<tr>
<td>float getPitch () const</td>
<td>Get the pitch of the sound. More...</td>
</tr>
<tr>
<td>float getVolume () const</td>
<td></td>
</tr>
</tbody>
</table>
**Vector3f getPosition() const**
Get the 3D position of the sound in the audio scene. More...

**bool isRelativeToListener() const**
Tell whether the sound's position is relative to the listener or absolute.

**float getMinDistance() const**
Get the minimum distance of the sound. More...

**float getAttenuation() const**
Get the attenuation factor of the sound. More...

**SoundSource & operator=(const SoundSource &right)**
Overload of assignment operator. More...
## Protected Member Functions

**SoundSource** ()
Default constructor. More...

**Status**
**getStatus** () const
Get the current status of the sound (stopped, paused, playing) More...
## Protected Attributes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>m_source</td>
<td>OpenAL source identifier.</td>
</tr>
</tbody>
</table>
Detailed Description

Base class defining a sound's properties.

sf::SoundSource is not meant to be used directly, it only serves as a common base for all audio objects that can live in the audio environment.

It defines several properties for the sound: pitch, volume, position, attenuation, etc. All of them can be changed at any time with no impact on performances.

See also

sf::Sound, sf::SoundStream

Definition at line 42 of file SoundSource.hpp.
### enum sf::SoundSource::Status

Enumeration of the sound source states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopped</td>
<td>Sound is not playing.</td>
</tr>
<tr>
<td>Paused</td>
<td>Sound is paused.</td>
</tr>
<tr>
<td>Playing</td>
<td>Sound is playing.</td>
</tr>
</tbody>
</table>

Definition at line 50 of file SoundSource.hpp.
Constructor & Destructor Documentation

sf::SoundSource::SoundSource ( const SoundSource & copy )

Copy constructor.

**Parameters**

*copy* Instance to copy

virtual sf::SoundSource::~SoundSource ( )

Destructor.

sf::SoundSource::SoundSource ( )

Default constructor.

This constructor is meant to be called by derived classes only.
Member Function Documentation

float sf::SoundSource::getAttenuation ( ) const

Get the attenuation factor of the sound.

Returns
   Attenuation factor of the sound

See also
   setAttenuation, getMinDistance

float sf::SoundSource::getMinDistance ( ) const

Get the minimum distance of the sound.

Returns
   Minimum distance of the sound

See also
   setMinDistance, getAttenuation

float sf::SoundSource::getPitch ( ) const

Get the pitch of the sound.
Returns
  Pitch of the sound

See also
  setPitch

**Vector3f** sf::SoundSource::getPosition ( ) const

Get the 3D position of the sound in the audio scene.

**Returns**
  Position of the sound

See also
  setPosition

**Status** sf::SoundSource::getStatus ( ) const

Get the current status of the sound (stopped, paused, playing)

**Returns**
  Current status of the sound

**float** sf::SoundSource::getVolume ( ) const

Get the volume of the sound.

**Returns**
  Volume of the sound, in the range [0, 100]
See also
   setVolume

bool sf::SoundSource::isRelativeToListener () const

Tell whether the sound's position is relative to the listener or is absolute.

Returns
   True if the position is relative, false if it's absolute

See also
   setRelativeToListener

SoundSource& sf::SoundSource::operator= ( const SoundSource & right)

Overload of assignment operator.

Parameters
   right Instance to assign

Returns
   Reference to self

void sf::SoundSource::setAttenuation ( float attenuation )

Set the attenuation factor of the sound.

The attenuation is a multiplicative factor which makes the sound more or less loud according to its distance from the listener. An attenuation of 0 will produce a non-atten
always be the same whether it is heard from near or from far. On the other hand, an attenuation value such as 100 will make the sound fade out very quickly as it gets further from the listener. The default value of the attenuation is 1.

**Parameters**

- **attenuation** New attenuation factor of the sound

**See also**

- `getAttenuation`, `setMinDistance`

---

**void sf::SoundSource::setMinDistance ( float distance )**

Set the minimum distance of the sound.

The "minimum distance" of a sound is the maximum distance at which it is heard at its maximum volume. Further than the minimum distance, it will start to fade out according to its attenuation factor. A value of 0 ("inside the head of the listener") is an invalid value and is forbidden. The default value of the minimum distance is 1.

**Parameters**

- **distance** New minimum distance of the sound

**See also**

- `getMinDistance`, `setAttenuation`

---

**void sf::SoundSource::setPitch ( float pitch )**

Set the pitch of the sound.

The pitch represents the perceived fundamental frequency of a sound; thus you can make a sound more acute or grave by changing its pitch. A side effect of changing the pitch is to modify the playing speed of
the sound as well. The default value for the pitch is 1.

**Parameters**

`pitch` New pitch to apply to the sound

**See also**

`getPitch`

---

### `void sf::SoundSource::setPosition ( float x, float y, float z)`

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position of a sound is (0, 0, 0).

**Parameters**

- `x` X coordinate of the position of the sound in the scene
- `y` Y coordinate of the position of the sound in the scene
- `z` Z coordinate of the position of the sound in the scene

**See also**

`getPosition`

---

### `void sf::SoundSource::setPosition ( const Vector3f & position )`

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position of a sound is (0, 0, 0).
Parameters

**position** Position of the sound in the scene

See also

ggetPosition

---

**void sf::SoundSource::setRelativeToListener ( bool relative )**

Make the sound's position relative to the listener or absolute.

Making a sound relative to the listener will ensure that it will always be balanced with the position of the listener. This can be useful for non-spatialized sounds, like the listener, or sounds attached to it. The default value is false (position is absolute).

Parameters

**relative** True to set the position relative, false to set it absolute

See also

isRelativeToListener

---

**void sf::SoundSource::setVolume ( float volume )**

Set the volume of the sound.

The volume is a value between 0 (mute) and 100 (full volume). The default value is 100.

Parameters

**volume** Volume of the sound
See also

getVolume
Member Data Documentation

**unsigned int sf::SoundSource::m_source**

OpenAL source identifier.

Definition at line 274 of file `SoundSource.hpp`.

The documentation for this class was generated from the following file:
- `SoundSource.hpp`

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila :: Documentation generated by doxygen
### sf::SoundStream Class Reference

**Audio module**

Abstract base class for streamed audio sources. More...

```cpp
#include <SoundStream.hpp>
```

Inheritance diagram for sf::SoundStream:
# Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>Chunk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure defining a chunk of audio data to stream. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Public Types

```cpp
enum Status { Stopped, Paused, Playing }
Enumeration of the sound source states. More...
```
## Public Member Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual</td>
<td><code>~SoundStream()</code></td>
<td>Destructor. More...</td>
</tr>
<tr>
<td>void</td>
<td><code>play()</code></td>
<td>Start or resume playing the audio stream. More...</td>
</tr>
<tr>
<td>void</td>
<td><code>pause()</code></td>
<td>Pause the audio stream. More...</td>
</tr>
<tr>
<td>void</td>
<td><code>stop()</code></td>
<td>Stop playing the audio stream. More...</td>
</tr>
<tr>
<td>unsigned int</td>
<td><code>getChannelCount()</code></td>
<td>Return the number of channels of the stream. More...</td>
</tr>
<tr>
<td>unsigned int</td>
<td><code>getSampleRate()</code></td>
<td>Get the stream sample rate of the stream. More...</td>
</tr>
<tr>
<td>Status</td>
<td><code>getStatus()</code></td>
<td>Get the current status of the stream (stopped, paused, playing)</td>
</tr>
<tr>
<td>void</td>
<td><code>setPlayingOffset(Time timeOffset)</code></td>
<td>Change the current playing position of the stream. More...</td>
</tr>
<tr>
<td>Time</td>
<td><code>getPlayingOffset()</code></td>
<td>Get the current playing position of the stream. More...</td>
</tr>
<tr>
<td>void</td>
<td><code>setLoop(bool loop)</code></td>
<td>Set whether or not the stream should loop after reaching the end. More...</td>
</tr>
<tr>
<td>bool</td>
<td><code>getLoop()</code></td>
<td></td>
</tr>
</tbody>
</table>
Tell whether or not the stream is in loop mode. More...

```cpp
void setPitch (float pitch)
Set the pitch of the sound. More...
```

```cpp
void setVolume (float volume)
Set the volume of the sound. More...
```

```cpp
void setPosition (float x, float y, float z)
Set the 3D position of the sound in the audio scene. More..
```

```cpp
void setPosition (const Vector3f &position)
Set the 3D position of the sound in the audio scene. More..
```

```cpp
void setRelativeToListener (bool relative)
Make the sound's position relative to the listener or absolute.
```

```cpp
void setMinDistance (float distance)
Set the minimum distance of the sound. More...
```

```cpp
void setAttenuation (float attenuation)
Set the attenuation factor of the sound. More...
```

```cpp
float getPitch () const
Get the pitch of the sound. More...
```

```cpp
float getVolume () const
Get the volume of the sound. More...
```

```cpp
Vector3f getPosition () const
Get the 3D position of the sound in the audio scene. More..
```

```cpp
bool isRelativeToListener () const
Tell whether the sound's position is relative to the listener or absolute.
```
float `getMinDistance ()` const
Get the minimum distance of the sound. More...

float `getAttenuation ()` const
Get the attenuation factor of the sound. More...
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoundStream ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void initialize (unsigned int channelCount, unsigned int sampleRate)</strong></td>
<td>Define the audio stream parameters. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>virtual bool onGetData (Chunk &amp;data)=0</strong></td>
<td>Request a new chunk of audio samples from the stream source.</td>
</tr>
<tr>
<td><strong>virtual void onSeek (Time timeOffset)=0</strong></td>
<td>Change the current playing position in the stream source. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Protected Attributes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>m_source</td>
<td>OpenAL source identifier.</td>
</tr>
</tbody>
</table>

More...
Detailed Description

Abstract base class for streamed audio sources.

Unlike audio buffers (see `sf::SoundBuffer`), audio streams are never completely loaded in memory.

Instead, the audio data is acquired continuously while the stream is playing, allowing a sound with no loading delay, and keeps the memory consumption very low.

Sound sources that need to be streamed are usually big files (compressed audio musics that would eat hundreds of MB in memory) or files that would take a lot of time to be received over the network.

`sf::SoundStream` is a base class that doesn't care about the stream source, which is left to the derived class. SFML provides a built-in specialization for big files (see `sf::Music` provided, but you can write your own by combining this class with the network module).

A derived class has to override two virtual functions:

- `onGetData` fills a new chunk of audio data to be played
- `onSeek` changes the current playing position in the source

It is important to note that each `SoundStream` is played in its own separate thread, so the loop doesn't block the rest of the program. In particular, the `OnGetData` and `OnSeek` functions may sometimes be called from this separate thread. It is important to keep this in mind, as you may have to take care of synchronization issues if you share data between threads.

Usage example:

```cpp
class CustomStream : public sf::SoundStream
{
    public:
```
bool open(const std::string& location)
{
  // Open the source and get audio settings
  ...
  unsigned int channelCount = ...;
  unsigned int sampleRate = ...;

  // Initialize the stream -- important!
  initialize(channelCount, sampleRate);
}

private:
  virtual bool onGetData(Chunk& data)
  {
    // Fill the chunk with audio data from the stream source
    // (note: must not be empty if you want to continue playing)
    data.samples = ...;
    data.sampleCount = ...;

    // Return true to continue playing
    return true;
  }

  virtual void onSeek(Uint32 timeOffset)
  {
    // Change the current position in the stream source
    ...
  }

  // Usage
  CustomStream stream;
  stream.open("path/to/stream");
  stream.play();

See also
  sf::Music

Definition at line 45 of file SoundStream.hpp.
**Member Enumeration Documentation**

**enum sf::SoundSource::Status**

Enumeration of the sound source states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopped</td>
<td>Sound is not playing.</td>
</tr>
<tr>
<td>Paused</td>
<td>Sound is paused.</td>
</tr>
<tr>
<td>Playing</td>
<td>Sound is playing.</td>
</tr>
</tbody>
</table>

Definition at line 50 of file `SoundSource.hpp`. 
Constructor & Destructor Documentation

**virtual sf::SoundStream::~SoundStream ( )**

Destructor.

**sf::SoundStream::SoundStream ( )**

Default constructor.

This constructor is only meant to be called by derived classes.
# Member Function Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `float sf::SoundSource::getAttenuation ( ) const` | Get the attenuation factor of the sound.  
**Returns**  
Attenuation factor of the sound  
**See also**  
`setAttenuation, getMinDistance` |
| `unsigned int sf::SoundStream::getChannelCount ( ) const` | Return the number of channels of the stream.  
1 channel means a mono sound, 2 means stereo, etc.  
**Returns**  
Number of channels |
| `bool sf::SoundStream::getLoop ( ) const` | Tell whether or not the stream is in loop mode.  
**Returns** |
True if the stream is looping, false otherwise

See also
setLoop

float sf::SoundSource::getMinDistance ( ) const

Get the minimum distance of the sound.

Returns
Minimum distance of the sound

See also
setMinDistance, getAttenuation

float sf::SoundSource::getPitch ( ) const

Get the pitch of the sound.

Returns
Pitch of the sound

See also
setPitch

Time sf::SoundStream::getPlayingOffset ( ) const

Get the current playing position of the stream.

Returns
Current playing position, from the beginning of the stream

See also
setPlayingOffset

**Vector3f sf::SoundSource::getPosition ( ) const**

Get the 3D position of the sound in the audio scene.

**Returns**
Position of the sound

See also
setPosition

**unsigned int sf::SoundStream::getSampleRate ( ) const**

Get the stream sample rate of the stream.

The sample rate is the number of audio samples played per second. The

**Returns**
Sample rate, in number of samples per second

**Status sf::SoundStream::getStatus ( ) const**

Get the current status of the stream (stopped, paused, playing)

**Returns**
Current status
float sf::SoundSource::getVolume ( ) const

Get the volume of the sound.

Returns

Volume of the sound, in the range [0, 100]

See also

setVolume

void sf::SoundStream::initialize ( unsigned int channelCount, unsigned int sampleRate )

Define the audio stream parameters.

This function must be called by derived classes as soon as they know the play. Any attempt to manipulate the stream (play(), ...) before calling this multiple times if the settings of the audio stream change, but only when t

Parameters

channelCount Number of channels of the stream
sampleRate Sample rate, in samples per second

bool sf::SoundSource::isRelativeToListener ( ) const

Tell whether the sound's position is relative to the listener or is absolute.
virtual bool sf::SoundStream::onGetData ( Chunk & data )

Request a new chunk of audio samples from the stream source.

This function must be overridden by derived classes to provide the audio samples continuously by the streaming loop, in a separate thread. The source can stop the streaming loop at any time, by returning false to the caller. If you return true (i.e. continue streaming) it is important that the returned array of samples is not empty; this would stop the stream.

Parameters

- **data** Chunk of data to fill

Returns

- True to continue playback, false to stop

Implemented in sf::Music.

virtual void sf::SoundStream::onSeek ( Time timeOffset )

Change the current playing position in the stream source.

This function must be overridden by derived classes to allow random seeking into the stream source.

Parameters

- **timeOffset** New playing position, relative to the beginning of the stream.
Implemented in sf::Music.

```cpp
void sf::SoundStream::pause()
```

Pause the audio stream.

This function pauses the stream if it was playing, otherwise (stream already paused or stopped) it has no effect.

**See also**
- play, stop

```cpp
void sf::SoundStream::play()
```

Start or resume playing the audio stream.

This function starts the stream if it was stopped, resumes it if it was paused, and restarts it from the beginning if it was already playing. This function uses its own thread so that it doesn’t block the rest of the program while the stream is played.

**See also**
- pause, stop

```cpp
void sf::SoundSource::setAttenuation(float attenuation)
```

Set the attenuation factor of the sound.

The attenuation is a multiplicative factor which makes the sound more or less loud according to its distance from the listener. An attenuation of 0 will produce a non-atten
always be the same whether it is heard from near or from far. On the other hand, an attenuation value such as 100 will make the sound fade out very quickly as it gets further away. The default value of the attenuation is 1.

**Parameters**

| attenuation | New attenuation factor of the sound |

**See also**

- getAttenuation, setMinDistance

---

**void sf::SoundStream::setLoop ( bool loop )**

Set whether or not the stream should loop after reaching the end.

If set, the stream will restart from the beginning after reaching the end and so on, until `setLoop(false)` is called. The default looping state for streams is false.

**Parameters**

| loop | True to play in loop, false to play once |

**See also**

- getLoop

---

**void sf::SoundSource::setMinDistance ( float distance )**

Set the minimum distance of the sound.

The "minimum distance" of a sound is the maximum distance at which it can be heard. Further than the minimum distance, it will start to fade out according to its attenuation factor. A value of 0 ("inside the head of the listener") is an invalid value and is forbidden. The default value of the minimum distance is 1.
Parameters
distance New minimum distance of the sound

See also
getMinDistance, setAttenuation

void sf::SoundSource::setPitch ( float pitch )

Set the pitch of the sound.

The pitch represents the perceived fundamental frequency of a sound; the sound can be made more acute or grave by changing its pitch. A side effect of changing the pitch is that it will also alter the playing speed of the sound as well. The default value for the pitch is 1.

Parameters
pitch New pitch to apply to the sound

See also
getPitch

void sf::SoundStream::setPlayingOffset ( Time timeOffset )

Change the current playing position of the stream.

The playing position can be changed when the stream is either paused or playing. Changing the playing position when the stream is stopped has no effect, since playing the stream would reset its position.

Parameters
timeOffset New playing position, from the beginning of the stream
See also
   getPlayingOffset

```
void sf::SoundSource::setPosition ( float x,  
     float y,  
     float z  
 )
```

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position of a sound is (0, 0, 0).

**Parameters**

- **x** X coordinate of the position of the sound in the scene
- **y** Y coordinate of the position of the sound in the scene
- **z** Z coordinate of the position of the sound in the scene

See also
   getPosition

```
void sf::SoundSource::setPosition ( const Vector3f & position )
```

Set the 3D position of the sound in the audio scene.

Only sounds with one channel (mono sounds) can be spatialized. The default position is (0, 0, 0).

**Parameters**

- **position** Position of the sound in the scene
See also
getPosition

**void sf::SoundSource::setRelativeToListener ( bool relative )**

Make the sound's position relative to the listener or absolute.

Making a sound relative to the listener will ensure that it will always be relative to the position of the listener. This can be useful for non-spatialized sounds, or sounds produced by the listener, or sounds attached to it. The default value is false (position is absolute).

**Parameters**
relative True to set the position relative, false to set it absolute

See also
isRelativeToListener

**void sf::SoundSource::setVolume ( float volume )**

Set the volume of the sound.

The volume is a value between 0 (mute) and 100 (full volume). The default value for the volume is 100.

**Parameters**
volume Volume of the sound

See also
getVolume

**void sf::SoundStream::stop ( )**
Stop playing the audio stream.

This function stops the stream if it was playing or paused, and does not also resets the playing position (unlike `pause()`).

**See also**

`play`, `pause`
Member Data Documentation

**unsigned int sf::SoundSource::m_source**

OpenAL source identifier.

Definition at line 274 of file `SoundSource.hpp`.

The documentation for this class was generated from the following file:
- `SoundStream.hpp`

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila  ::  Documentation generated by doxygen
sf::SoundStream::Chunk Struct Reference

Structure defining a chunk of audio data to stream. More...

#include <SoundStream.hpp>
# Public Attributes

<table>
<thead>
<tr>
<th>const Int16 * samples</th>
<th>Pointer to the audio samples. More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>std::size_t sampleCount</td>
<td>Number of samples pointed by Samples. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Structure defining a chunk of audio data to stream.

Definition at line 53 of file SoundStream.hpp.
Member Data Documentation

<table>
<thead>
<tr>
<th>std::size_t sf::SoundStream::Chunk::sampleCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of samples pointed by Samples.</td>
</tr>
<tr>
<td>Definition at line 56 of file SoundStream.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>const Int16* sf::SoundStream::Chunk::samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the audio samples.</td>
</tr>
<tr>
<td>Definition at line 55 of file SoundStream.hpp.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:
- SoundStream.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen
Namespaces | Classes | Enumerations

Graphics module

2D graphics module: sprites, text, shapes, ... More...
Namespaces

sf::Glsl
Namespace with GLSL types.
## Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::BlendMode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Blending modes for drawing. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::CircleShape</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specialized shape representing a circle. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::Color</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utility class for manipulating RGBA colors. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::ConvexShape</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specialized shape representing a convex polygon. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::Drawable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abstract base class for objects that can be drawn to a render target</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::Font</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class for loading and manipulating character fonts. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure describing a glyph. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::Image</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class for loading, manipulating and saving images. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::Rect&lt; T &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utility class for manipulating 2D axis aligned rectangles. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::RectangleShape</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specialized shape representing a rectangle. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>sf::RenderStates</th>
</tr>
</thead>
</table>
Define the states used for drawing to a **RenderTarget**. More...

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>sf::RenderTarget</em></td>
<td>Base class for all render targets (window, texture, ...) More...</td>
</tr>
<tr>
<td><em>sf::RenderTexture</em></td>
<td>Target for off-screen 2D rendering into a texture. More...</td>
</tr>
<tr>
<td><em>sf::RenderWindow</em></td>
<td>Window that can serve as a target for 2D drawing. More...</td>
</tr>
<tr>
<td><em>sf::Shader</em></td>
<td>Shader class (vertex, geometry and fragment) More...</td>
</tr>
<tr>
<td><em>sf::Shape</em></td>
<td>Base class for textured shapes with outline. More...</td>
</tr>
<tr>
<td><em>sf::Sprite</em></td>
<td>Drawable representation of a texture, with its own transformations</td>
</tr>
<tr>
<td><em>sf::Text</em></td>
<td>Graphical text that can be drawn to a render target. More...</td>
</tr>
<tr>
<td><em>sf::Texture</em></td>
<td>Image living on the graphics card that can be used for drawing. More...</td>
</tr>
<tr>
<td><em>sf::Transform</em></td>
<td>Define a 3x3 transform matrix. More...</td>
</tr>
<tr>
<td><em>sf::Transformable</em></td>
<td>Decomposed transform defined by a position, a rotation and a scale</td>
</tr>
<tr>
<td><em>sf::Vertex</em></td>
<td>Define a point with color and texture coordinates. More...</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>sf::VertexArray</strong></td>
<td>Define a set of one or more 2D primitives. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>sf::View</strong></td>
<td>2D camera that defines what region is shown on screen <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Enumerations

```cpp
enum sf::PrimitiveType {
    sf::Points, sf::Lines, sf::LineStrip, sf::Triangles,
    sf::TriangleStrip, sf::TriangleFan, sf::Quads, sf::LinesStrip = Line,
    sf::TrianglesStrip = TriangleStrip, sf::TrianglesFan = TriangleFan
}
```

Types of primitives that a `sf::VertexArray` can render. More...
Detailed Description

2D graphics module: sprites, text, shapes, ...
# Enumeration Type Documentation

**enum sf::PrimitiveType**

Types of primitives that a *sf::VertexArray* can render.

Points and lines have no area, therefore their thickness will always be 1 pixel, regardless of the current transform and view.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>List of individual points.</td>
</tr>
<tr>
<td>Lines</td>
<td>List of individual lines.</td>
</tr>
<tr>
<td>LineStrip</td>
<td>List of connected lines, a point uses the previous point.</td>
</tr>
<tr>
<td>Triangles</td>
<td>List of individual triangles.</td>
</tr>
<tr>
<td>TriangleStrip</td>
<td>List of connected triangles, a point uses the two previous points.</td>
</tr>
<tr>
<td>TriangleFan</td>
<td>List of connected triangles, a point uses the common center and the previous point to form a triangle.</td>
</tr>
<tr>
<td>Quads</td>
<td>List of individual quads (deprecated, don't work with OpenGL ES).</td>
</tr>
<tr>
<td>LinesStrip</td>
<td>Deprecated: Use LineStrip instead</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>TrianglesStrip</td>
<td>Deprecated: Use TriangleStrip instead</td>
</tr>
<tr>
<td>TrianglesFan</td>
<td>Deprecated: Use TriangleFan instead</td>
</tr>
</tbody>
</table>

Definition at line 39 of file PrimitiveType.hpp.

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
typedefs

sf::Glsl Namespace Reference
Graphics module

Namespace with GLSL types. More...
### Typedefs

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Implementation</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef Vector2&lt; float &gt;</code></td>
<td></td>
<td>Vec2 2D float vector (<code>vec2</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef Vector2&lt; int &gt;</code></td>
<td></td>
<td>Ivec2 2D int vector (<code>ivec2</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef Vector2&lt; bool &gt;</code></td>
<td></td>
<td>Bvec2 2D bool vector (<code>bvec2</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef Vector3&lt; float &gt;</code></td>
<td></td>
<td>Vec3 3D float vector (<code>vec3</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef Vector3&lt; int &gt;</code></td>
<td></td>
<td>Ivec3 3D int vector (<code>ivec3</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef Vector3&lt; bool &gt;</code></td>
<td></td>
<td>Bvec3 3D bool vector (<code>bvec3</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef implementation defined</code></td>
<td>Vec4</td>
<td>4D float vector (<code>vec4</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef implementation defined</code></td>
<td>Ivec4</td>
<td>4D int vector (<code>ivec4</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef implementation defined</code></td>
<td>Bvec4</td>
<td>4D bool vector (<code>bvec4</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef implementation defined</code></td>
<td>Mat3</td>
<td>3x3 float matrix (<code>mat3</code> in GLSL)</td>
<td></td>
</tr>
<tr>
<td><code>typedef implementation defined</code></td>
<td>Mat4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4x4 float matrix (`mat4` in GLSL)
Detailed Description

Namespace with GLSL types.

The `sf::Glsl` namespace contains types that match their equivalents in GLSL, the OpenGL shading language. These types are exclusively used by the `sf::Shader` class.

Types that already exist in SFML, such as `sf::Vector2<T>` and `sf::Vector3<T>`, you can use the types in this namespace as well as the original ones. C `Glsl::Vec4` or `Glsl::Mat3`. Their actual type is an implementation detail and

All vector types support a default constructor that initializes every constructor with one parameter for each component. The components called `x, y, z,` and `w`.

All matrix types support a constructor with a `float*` parameter that points size (that is, 9 in a 3x3 matrix, 16 in a 4x4 matrix). Furthermore, they can objects.

**See also**

`sf::Shader`
## Typedef Documentation

### `typedef Vector2<bool> sf::Glsl::Bvec2`

2D bool vector (`bvec2` in GLSL)

Definition at line 76 of file `Glsl.hpp`.

### `typedef Vector3<bool> sf::Glsl::Bvec3`

3D bool vector (`bvec3` in GLSL)

Definition at line 94 of file `Glsl.hpp`.

### `typedef implementation defined sf::Glsl::Bvec4`

4D bool vector (`bvec4` in GLSL)

Definition at line 130 of file `Glsl.hpp`.

### `typedef Vector2<int> sf::Glsl::Ivec2`

2D int vector (`ivec2` in GLSL)
### typedef Vector3<int> sf::Glsl::Ivec3

3D int vector (ivec3 in GLSL)

Definition at line 70 of file Glsl.hpp.

### typedef implementation defined sf::Glsl::Ivec4

4D int vector (ivec4 in GLSL)

4D int vectors can be implicitly converted from sf::Color instances unchanged inside the integer interval [0, 255].

```cpp
sf::Glsl::Ivec4 zeroVector;
sf::Glsl::Ivec4 vector(1, 2, 3, 4);
sf::Glsl::Ivec4 color = sf::Color::Cyan;
```

Definition at line 124 of file Glsl.hpp.

### typedef implementation defined sf::Glsl::Mat3

3x3 float matrix (mat3 in GLSL)

The matrix can be constructed from an array with 3x3 elements, aligned example, a translation by (x, y) looks as follows:

```cpp
float array[9] =
{
    1, 0, 0,
```
Mat3 can also be implicitly converted from `sf::Transform`:

```cpp
sf::Transform transform;
sf::Glsl::Mat3 matrix = transform;
```

Definition at line 155 of file `Glsl.hpp`.

typedef implementation defined `sf::Glsl::Mat4`

4x4 float matrix (`mat4` in GLSL)

The matrix can be constructed from an array with 4x4 elements, aligned as follows:

```cpp
float array[16] =
{ 1, 0, 0, 0,
  0, 1, 0, 0,
  0, 0, 1, 0,
  x, y, z, 1
};
sf::Glsl::Mat4 matrix(array);
```

Mat4 can also be implicitly converted from `sf::Transform`:

```cpp
sf::Transform transform;
sf::Glsl::Mat4 matrix = transform;
```

Definition at line 181 of file `Glsl.hpp`. 
### typedef Vector2<float> sf::Glsl::Vec2

2D float vector (`vec2` in GLSL)

Definition at line 64 of file `Glsl.hpp`.

### typedef Vector3<float> sf::Glsl::Vec3

3D float vector (`vec3` in GLSL)

Definition at line 82 of file `Glsl.hpp`.

### typedef implementation defined sf::Glsl::Vec4

4D float vector (`vec4` in GLSL)

4D float vectors can be implicitly converted from `sf::Color` instances. Each color channel is normalized from integers in [0, 255] to floating point values in [0, 1].

```cpp
sf::Glsl::Vec4 zeroVector;
sf::Glsl::Vec4 vector(1.f, 2.f, 3.f, 4.f);
sf::Glsl::Vec4 color = sf::Color::Cyan;
```

Definition at line 110 of file `Glsl.hpp`.

---

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila  ::  Documentation generated by doxygen
sf::BlendMode Class Reference

Graphics module

Blending modes for drawing. More...

#include <BlendMode.hpp>
### Public Types

<table>
<thead>
<tr>
<th>enum</th>
<th>Factor { Zero, One, SrcColor, OneMinusSrcColor, DstColor, OneMinusDstColor, SrcAlpha, OneMinusSrcAlpha, DstAlpha, OneMinusDstAlpha }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enumeration of the blending factors. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>enum</th>
<th>Equation { Add, Subtract, ReverseSubtract }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enumeration of the blending equations. More...</td>
</tr>
</tbody>
</table>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BlendMode</strong> ()</td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>BlendMode</strong> (Factor sourceFactor, Factor destinationFactor, Equation blendEquation)</td>
<td>Construct the blend mode given the factors and equation. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>BlendMode</strong> (Factor colorSourceFactor, Factor colorDestinationFactor, Factor alphaSourceFactor, Factor alphaDestinationFactor, Equation alphaBlendEquation)</td>
<td>Construct the blend mode given the factors and equation. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor</td>
<td>colorSrcFactor</td>
<td>Source blending factor for the color channels. More...</td>
</tr>
<tr>
<td>Factor</td>
<td>colorDstFactor</td>
<td>Destination blending factor for the color channels. More...</td>
</tr>
<tr>
<td>Equation</td>
<td>colorEquation</td>
<td>Blending equation for the color channels. More...</td>
</tr>
<tr>
<td>Factor</td>
<td>alphaSrcFactor</td>
<td>Source blending factor for the alpha channel. More...</td>
</tr>
<tr>
<td>Factor</td>
<td>alphaDstFactor</td>
<td>Destination blending factor for the alpha channel. More...</td>
</tr>
<tr>
<td>Equation</td>
<td>alphaEquation</td>
<td>Blending equation for the alpha channel. More...</td>
</tr>
</tbody>
</table>
Related Functions

(Note that these are not member functions.)

```cpp
bool operator==(const BlendMode &left, const BlendMode &right)
Overload of the == operator. More...
```

```cpp
bool operator!=(const BlendMode &left, const BlendMode &right)
Overload of the != operator. More...
```
Detailed Description

Blending modes for drawing.

sf::BlendMode is a class that represents a blend mode.

A blend mode determines how the colors of an object you draw are mixed with the buffer.

The class is composed of 6 components, each of which has its own public:

- Color Source Factor (colorSrcFactor)
- Color Destination Factor (colorDstFactor)
- Color Blend Equation (colorEquation)
- Alpha Source Factor (alphaSrcFactor)
- Alpha Destination Factor (alphaDstFactor)
- Alpha Blend Equation (alphaEquation)

The source factor specifies how the pixel you are drawing contributes to the final color.

The destination factor specifies how the pixel already drawn in the buffer contributes to the final color.

The color channels RGB (red, green, blue; simply referred to as color) and A (alpha; the transparency) can be treated separately. This separation can be useful for specific blend modes, but most often you won't need it and will simply treat the color as a single unit.

The blend factors and equations correspond to their OpenGL equivalents. The resulting pixel is calculated according to the following formula (src is the color of the source pixel, the other variables correspond to the public members, being + or - operators):
dst.rgb = colorSrcFactor * src.rgb (colorEquation) colorDstFactor * dst.rgb
dst.a = alphaSrcFactor * src.a (alphaEquation) alphaDstFactor * dst.a

All factors and colors are represented as floating point numbers between 0 and 1. The result is clamped to fit in that range.

The most common blending modes are defined as constants in the sf namespace:

<table>
<thead>
<tr>
<th>sf::BlendMode</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphaBlending</td>
<td>sf::BlendAlpha</td>
</tr>
<tr>
<td>additiveBlending</td>
<td>sf::BlendAdd</td>
</tr>
<tr>
<td>multiplicativeBlending</td>
<td>sf::BlendMultiply</td>
</tr>
<tr>
<td>noBlending</td>
<td>sf::BlendNone</td>
</tr>
</tbody>
</table>

In SFML, a blend mode can be specified every time you draw a sf::Drawable part of the sf::RenderStates compound that is passed to the member function.

See also
sf::RenderStates, sf::RenderTarget

Definition at line 41 of file BlendMode.hpp.
Member Enumeration Documentation

**enum sf::BlendMode::Equation**

Enumeration of the blending equations.

The equations are mapped directly to their OpenGL equivalents, `glBlendEquation()` or `glBlendEquationSeparate()`.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong></td>
<td>$\text{Pixel} = \text{Src} \times \text{SrcFactor} + \text{Dst} \times \text{DstFactor}$.</td>
</tr>
<tr>
<td><strong>Subtract</strong></td>
<td>$\text{Pixel} = \text{Src} \times \text{SrcFactor} - \text{Dst} \times \text{DstFactor}$.</td>
</tr>
<tr>
<td><strong>ReverseSubtract</strong></td>
<td>$\text{Pixel} = \text{Dst} \times \text{DstFactor} - \text{Src} \times \text{SrcFactor}$.</td>
</tr>
</tbody>
</table>

Definition at line 69 of file `BlendMode.hpp`.

**enum sf::BlendMode::Factor**

Enumeration of the blending factors.

The factors are mapped directly to their OpenGL equivalents, `glBlendFunc()` or `glBlendFuncSeparate()`.
<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>(0, 0, 0, 0)</td>
</tr>
<tr>
<td>One</td>
<td>(1, 1, 1, 1)</td>
</tr>
<tr>
<td>SrcColor</td>
<td>(src.r, src.g, src.b, src.a)</td>
</tr>
<tr>
<td>OneMinusSrcColor</td>
<td>(1, 1, 1, 1) - (src.r, src.g, src.b, src.a)</td>
</tr>
<tr>
<td>DstColor</td>
<td>(dst.r, dst.g, dst.b, dst.a)</td>
</tr>
<tr>
<td>OneMinusDstColor</td>
<td>(1, 1, 1, 1) - (dst.r, dst.g, dst.b, dst.a)</td>
</tr>
<tr>
<td>SrcAlpha</td>
<td>(src.a, src.a, src.a, src.a)</td>
</tr>
<tr>
<td>OneMinusSrcAlpha</td>
<td>(1, 1, 1, 1) - (src.a, src.a, src.a, src.a)</td>
</tr>
<tr>
<td>DstAlpha</td>
<td>(dst.a, dst.a, dst.a, dst.a)</td>
</tr>
<tr>
<td>OneMinusDstAlpha</td>
<td>(1, 1, 1, 1) - (dst.a, dst.a, dst.a, dst.a)</td>
</tr>
</tbody>
</table>

Definition at line 49 of file BlendMode.hpp.
sf::BlendMode::BlendMode ( )

Default constructor.
Constructs a blending mode that does alpha blending.

sf::BlendMode::BlendMode ( Factor sourceFactor,
                          Factor destinationFactor,
                          Equation blendEquation = Add
                      )

Construct the blend mode given the factors and equation.
This constructor uses the same factors and equation for both color
defaults to the Add equation.

Parameters
  sourceFactor    Specifies how to compute the source factor for the
  destinationFactor Specifies how to compute the destination factor
                      for both color and alpha channels.
  blendEquation   Specifies how to combine the source and destination

sf::BlendMode::BlendMode ( Factor colorSourceFactor,
                          Factor colorDestinationFactor,
                          Equation blendEquation = Add
                      )
Construct the blend mode given the factors and equation.

**Parameters**
- `colorSourceFactor` Specifies how to compute the source factor for the color channels.
- `colorDestinationFactor` Specifies how to compute the destination factor for the color channels.
- `colorBlendEquation` Specifies how to combine the source and destination colors.
- `alphaSourceFactor` Specifies how to compute the source factor.
- `alphaDestinationFactor` Specifies how to compute the destination factor.
- `alphaBlendEquation` Specifies how to combine the source and destination alphas.
Friends And Related Function Documentation

bool operator!= (const BlendMode & left,
                const BlendMode & right
             )

Overload of the != operator.

Parameters
  - left  Left operand
  - right Right operand

Returns
  True if blending modes are different, false if they are equal

bool operator==(const BlendMode & left,
                 const BlendMode & right
              )

Overload of the == operator.

Parameters
  - left  Left operand
  - right Right operand

Returns
  True if blending modes are equal, false if they are different
Member Data Documentation

<table>
<thead>
<tr>
<th>Factor sf::BlendMode::alphaDstFactor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination blending factor for the alpha channel.</td>
</tr>
<tr>
<td>Definition at line 119 of file BlendMode.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation sf::BlendMode::alphaEquation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blending equation for the alpha channel.</td>
</tr>
<tr>
<td>Definition at line 120 of file BlendMode.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Factor sf::BlendMode::alphaSrcFactor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source blending factor for the alpha channel.</td>
</tr>
<tr>
<td>Definition at line 118 of file BlendMode.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Factor sf::BlendMode::colorDstFactor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination blending factor for the color channels.</td>
</tr>
<tr>
<td>Definition at line 116 of file BlendMode.hpp.</td>
</tr>
</tbody>
</table>
**Equation** sf::BlendMode::colorEquation

Blending equation for the color channels.

Definition at line 117 of file BlendMode.hpp.

**Factor** sf::BlendMode::colorSrcFactor

Source blending factor for the color channels.

Definition at line 115 of file BlendMode.hpp.

The documentation for this class was generated from the following file:
- BlendMode.hpp
sf::CircleShape Class Reference
Graphics module

Specialized shape representing a circle. More...

#include <CircleShape.hpp>

Inheritance diagram for sf::CircleShape:
### Public Member Functions

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CircleShape</td>
<td>(float radius=0, std::size_t pointCount=3)</td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>setRadius (float radius)</td>
<td>Set the radius of the circle. <a href="#">More...</a></td>
</tr>
<tr>
<td>float</td>
<td>getRadius () const</td>
<td>Get the radius of the circle. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>setPointCount (std::size_t count)</td>
<td>Set the number of points of the circle. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual std::size_t</td>
<td>getPointCount () const</td>
<td>Get the number of points of the circle. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual Vector2f</td>
<td>getPoint (std::size_t index) const</td>
<td>Get a point of the circle. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>setTexture (const Texture *texture, bool resetRect=false)</td>
<td>Change the source texture of the shape. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>setTextureRect (const IntRect &amp;rect)</td>
<td>Set the sub-rectangle of the texture that the shape will display.</td>
</tr>
<tr>
<td>void</td>
<td>setFillColor (const Color &amp;color)</td>
<td>Set the fill color of the shape. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>setOutlineColor (const Color &amp;color)</td>
<td>Set the outline color of the shape. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>setOutlineThickness (float thickness)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><code>const Texture * getTexture () const</code></td>
<td>Get the source texture of the shape.  More...</td>
<td></td>
</tr>
<tr>
<td><code>const IntRect &amp; getTextureRect () const</code></td>
<td>Get the sub-rectangle of the texture displayed by the shape.  More...</td>
<td></td>
</tr>
<tr>
<td><code>const Color &amp; getFillColor () const</code></td>
<td>Get the fill color of the shape.  More...</td>
<td></td>
</tr>
<tr>
<td><code>const Color &amp; getOutlineColor () const</code></td>
<td>Get the outline color of the shape.  More...</td>
<td></td>
</tr>
<tr>
<td><code>float getOutlineThickness () const</code></td>
<td>Get the outline thickness of the shape.  More...</td>
<td></td>
</tr>
<tr>
<td><code>FloatRect getLocalBounds () const</code></td>
<td>Get the local bounding rectangle of the entity.  More...</td>
<td></td>
</tr>
<tr>
<td><code>FloatRect getGlobalBounds () const</code></td>
<td>Get the global (non-minimal) bounding rectangle of the entity.  More...</td>
<td></td>
</tr>
<tr>
<td><code>void setPosition (float x, float y)</code></td>
<td>set the position of the object  More...</td>
<td></td>
</tr>
<tr>
<td><code>void setPosition (const Vector2f &amp;position)</code></td>
<td>set the position of the object  More...</td>
<td></td>
</tr>
<tr>
<td><code>void setRotation (float angle)</code></td>
<td>set the orientation of the object  More...</td>
<td></td>
</tr>
<tr>
<td><code>void setScale (float factorX, float factorY)</code></td>
<td>set the scale factors of the object  More...</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void setScale(const Vector2f &amp;factors)</td>
<td>set the scale factors of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>void setOrigin(float x, float y)</td>
<td>set the local origin of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>void setOrigin(const Vector2f &amp;origin)</td>
<td>set the local origin of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>const Vector2f &amp; getPosition() const</td>
<td>get the position of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>float getRotation() const</td>
<td>get the orientation of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>const Vector2f &amp; getScale() const</td>
<td>get the current scale of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>const Vector2f &amp; getOrigin() const</td>
<td>get the local origin of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>void move(float offsetX, float offsetY)</td>
<td>Move the object by a given offset. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>void move(const Vector2f &amp;offset)</td>
<td>Move the object by a given offset. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>void rotate(float angle)</td>
<td>Rotate the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>void scale(float offsetX, float offsetY)</td>
<td>Scale the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>void scale(const Vector2f &amp;factor)</td>
<td>Scale the object. <a href="#">More...</a></td>
<td></td>
</tr>
</tbody>
</table>
const Transform & getTransform () const
get the combined transform of the object

More...

const Transform & getInverseTransform () const
get the inverse of the combined transform of the object
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>update()</code></td>
<td><code>void</code></td>
<td>Recompute the internal geometry of the shape. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Specialized shape representing a circle.

This class inherits all the functions of `sf::Transformable` (position, rotation functions of `sf::Shape` (outline, color, texture, ...).

Usage example:

```cpp
sf::CircleShape circle;
circle.setRadius(150);
circle.setOutlineColor(sf::Color::Red);
circle.setOutlineThickness(5);
circle.setPosition(10, 20);
...
window.draw(circle);
```

Since the graphics card can't draw perfect circles, we have to fake them to each other. The "points count" property of `sf::CircleShape` defines how and therefore defines the quality of the circle.

The number of points can also be used for another purpose; with small numbers you can create any regular polygon shape: equilateral triangle, square, pentagon, hexagon, ...

**See also**

- `sf::Shape`, `sf::RectangleShape`, `sf::ConvexShape`

Definition at line 41 of file `CircleShape.hpp`. 
sf::CircleShape::CircleShape (float radius = 0, std::size_t pointCount = 30)

Default constructor.

Parameters

radius      Radius of the circle
pointCount  Number of points composing the circle
Member Function Documentation

**const Color**& sf::Shape::getFillColor ( ) const

Get the fill color of the shape.

**Returns**

Fill color of the shape

**See also**

setFillColor

**FloatRect** sf::Shape::getGlobalBounds ( ) const

Get the global (non-minimal) bounding rectangle of the entity.

The returned rectangle is in global coordinates, which means that transformations (translation, rotation, scale, ...) that are applied to the entity will be taken into account. This function returns the bounds of the shape in the global 2D world's coordinate system. This function does not necessarily return the *minimal* bounding rectangle. It merely ensures that the returned rectangle covers all the vertices (but possibly more). This allows for a fast approximation as a first check; you may want to use more precise checks on top.

**Returns**

Global bounding rectangle of the entity
**const Transform** & sf::Transformable::getInverseTransform ( ) const

get the inverse of the combined transform of the object

**Returns**
Inverse of the combined transformations applied to the object

**See also**
getTransform

**FloatRect** sf::Shape::getLocalBounds ( ) const

Get the local bounding rectangle of the entity.

The returned rectangle is in local coordinates, which means that (translation, rotation, scale, ...) that are applied to the entity. In other bounds of the entity in the entity's coordinate system.

**Returns**
Local bounding rectangle of the entity

**const Vector2f** & sf::Transformable::getOrigin ( ) const

get the local origin of the object

**Returns**
Current origin

**See also**
setOrigin
const Color& sf::Shape::getOutlineColor () const

Get the outline color of the shape.

Returns
Outline color of the shape

See also
setOutlineColor

float sf::Shape::getOutlineThickness () const

Get the outline thickness of the shape.

Returns
Outline thickness of the shape

See also
setOutlineThickness

virtual Vector2f sf::CircleShape::getPoint ( std::size_t index ) const

Get a point of the circle.

The returned point is in local coordinates, that is, the shape's transforms taken into account. The result is undefined if index is out of the valid range.

Parameters
index Index of the point to get, in range [0 .. getPointCount() - 1]
###>Returns
index-th point of the shape

Implements `sf::Shape`.

---

**virtual std::size_t sf::CircleShape::getPointCount ( ) const**

Get the number of points of the circle.

**Returns**
Number of points of the circle

**See also**
`setPointCount`

Implements `sf::Shape`.

---

**const Vector2f & sf::Transformable::getPosition ( ) const**

get the position of the object

**Returns**
Current position

**See also**
`setPosition`

---

**float sf::CircleShape::getRadius ( ) const**

Get the radius of the circle.
**Returns**

Radius of the circle

**See also**

setRadius

---

```cpp
float sf::Transformable::getRotation ( ) const
```

get the orientation of the object

The rotation is always in the range [0, 360].

**Returns**

Current rotation, in degrees

**See also**

setRotation

---

```cpp
const Vector2f& sf::Transformable::getScale ( ) const
```

get the current scale of the object

**Returns**

Current scale factors

**See also**

setScale

---

```cpp
const Texture* sf::Shape::getTexture ( ) const
```
Get the source texture of the shape.

If the shape has no source texture, a NULL pointer is returned. This means that you can't modify the texture when you retrieve it with this function.

**Returns**
- Pointer to the shape's texture

**See also**
- setTexture

```cpp
const IntRect& sf::Shape::getTextureRect() const
```

Get the sub-rectangle of the texture displayed by the shape.

**Returns**
- Texture rectangle of the shape

**See also**
- setTextureRect

```cpp
const Transform& sf::Transformable::getTransform() const
```

get the combined transform of the object

**Returns**
- Transform combining the position/rotation/scale/origin of the object

**See also**
- getInverseTransform
void sf::Transformable::move ( float offsetX, float offsetY )

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f pos = object.getPosition();
object.setPosition(pos.x + offsetX, pos.y + offsetY);
```

Parameters
- **offsetX** X offset
- **offsetY** Y offset

See also
- setPosition

---

void sf::Transformable::move ( const Vector2f & offset )

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition equivalent to the following code:

```cpp
object.setPosition(object.getPosition() + offset);
```

Parameters
- **offset** Offset
See also
setPosition

### void sf::Transformable::rotate ( float angle )

Rotate the object.

This function adds to the current rotation of the object, unlike setRotation which overwrites it. Thus, it is equivalent to the following code:

```
object.setRotation(object.getRotation() + angle);
```

**Parameters**
- **angle** Angle of rotation, in degrees

### void sf::Transformable::scale ( float factorX, float factorY )

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:

```
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factorX, scale.y * factorY);
```

**Parameters**
- **factorX** Horizontal scale factor
- **factorY** Vertical scale factor
See also
setScale

void sf::Transformable::scale ( const Vector2f & factor )

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factor.x, scale.y * factor.y);
```

Parameters

factor Scale factors

See also
setScale

See also

void sf::Shape::setFillColor ( const Color & color )

Set the fill color of the shape.

This color is modulated (multiplied) with the shape's texture if any. It can modulate or change its global opacity. You can use sf::Color::Transparent to render transparent, and have the outline alone. By default, the shape's fill color is white.

Parameters

color New color of the shape

See also
**void sf::Transformable::setOrigin(float x, float y)**

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object, and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

**Parameters**
- **x** X coordinate of the new origin
- **y** Y coordinate of the new origin

**See also**
- `getOrigin`

**void sf::Transformable::setOrigin(const Vector2f & origin)**

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object, and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

**Parameters**
- **origin** New origin

**See also**
void sf::Shape::setOutlineColor ( const Color & color )

Set the outline color of the shape.

By default, the shape's outline color is opaque white.

Parameters

color New outline color of the shape

See also
getOutlineColor, setFillColor

void sf::Shape::setOutlineThickness ( float thickness )

Set the thickness of the shape's outline.

Note that negative values are allowed (so that the outline expands towards the center of the shape), using zero disables the outline. By default, the outline thickness is 0.

Parameters

thickness New outline thickness

See also
getOutlineThickness

void sf::CircleShape::setPointCount ( std::size_t count )
Set the number of points of the circle.

**Parameters**
- **count** New number of points of the circle

**See also**
- `getPointCount`

```cpp
void sf::Transformable::setPosition ( float x, float y )
```

set the position of the object

This function completely overwrites the previous position. See the `move` function to apply an offset based on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**
- **x** X coordinate of the new position
- **y** Y coordinate of the new position

**See also**
- `move`, `getPosition`

```cpp
void sf::Transformable::setPosition ( const Vector2f & position )
```

set the position of the object

This function completely overwrites the previous position. See the `move` function to apply an offset based on the previous position instead. The default position of a transformable object is (0, 0).
Parameters

des Properties
New position

See also

to

getPosition

void sf::CircleShape::setRadius ( float radius )

Set the radius of the circle.

Parameters

radius New radius of the circle

See also

getRadius

void sf::Transformable::setRotation ( float angle )

set the orientation of the object

This function completely overwrites the previous rotation. See the rotate on the previous rotation instead. The default rotation of a transformable c

Parameters

angle New rotation, in degrees

See also

to, getRotation


```cpp
void sf::Transformable::setScale (float factorX, float factorY)
```

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function to add a factor based on the previous scale instead. The default scale of a transformable object is (1, 1).

**Parameters**
- `factorX` New horizontal scale factor
- `factorY` New vertical scale factor

**See also**
- `scale`, `getScale`

```cpp
void sf::Transformable::setScale (const Vector2f & factors)
```

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function to add a factor based on the previous scale instead. The default scale of a transformable object is (1, 1).

**Parameters**
- `factors` New scale factors

**See also**
- `scale`, `getScale`

```cpp
void sf::Shape::setTexture (const Texture * texture, bool resetRect = false)
```
Change the source texture of the shape.

The `texture` argument refers to a texture that must exist as long as the shape uses it. The shape doesn't store its own copy of the texture, but rather keeps a pointer to it. If the source texture is destroyed and the shape tries to use it, the behavior is undefined. The `texture` can be `NULL` to disable texturing. If `resetRect` is true, the `TextureRect` property of the shape is automatically adjusted to the size of the new texture. If it is false, the texture rect is left unchanged.

**Parameters**
- `texture` New texture
- `resetRect` Should the texture rect be reset to the size of the new texture?

**See also**
- `getTexture`, `setTextureRect`

```cpp
void sf::Shape::setTextureRect (const IntRect & rect)
```

Set the sub-rectangle of the texture that the shape will display.

The texture rect is useful when you don't want to display the whole texture, but rather a part of it. By default, the texture rect covers the entire texture.

**Parameters**
- `rect` Rectangle defining the region of the texture to display

**See also**
- `getTextureRect`, `setTexture`

```cpp
void sf::Shape::update ()
```
Recompute the internal geometry of the shape.

This function must be called by the derived class everytime the shape's either getPointCount or getPoint is different).

The documentation for this class was generated from the following file:
- CircleShape.hpp
sf::Color Class Reference

Graphics module

Utility class for manipulating RGBA colors. More...

#include <Color.hpp>
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Color ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Color (Uint8 red, Uint8 green, Uint8 blue, Uint8 alpha=255)</strong></td>
<td>Construct the color from its 4 RGBA components. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Color (Uint32 color)</strong></td>
<td>Construct the color from 32-bit unsigned integer. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Uint32 toInteger () const</strong></td>
<td>Retrieve the color as a 32-bit unsigned integer. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint8</td>
<td>r</td>
<td>Red component.</td>
<td>More</td>
</tr>
<tr>
<td>Uint8</td>
<td>g</td>
<td>Green component.</td>
<td>More</td>
</tr>
<tr>
<td>Uint8</td>
<td>b</td>
<td>Blue component.</td>
<td>More</td>
</tr>
<tr>
<td>Uint8</td>
<td>a</td>
<td>Alpha (opacity)</td>
<td>More</td>
</tr>
</tbody>
</table>
## Static Public Attributes

<table>
<thead>
<tr>
<th>static const Color</th>
<th>Black</th>
<th>Black predefined color. <a href="#">More...</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>static const Color</td>
<td>White</td>
<td>White predefined color. <a href="#">More...</a></td>
</tr>
<tr>
<td>static const Color</td>
<td>Red</td>
<td>Red predefined color. <a href="#">More...</a></td>
</tr>
<tr>
<td>static const Color</td>
<td>Green</td>
<td>Green predefined color. <a href="#">More...</a></td>
</tr>
<tr>
<td>static const Color</td>
<td>Blue</td>
<td>Blue predefined color. <a href="#">More...</a></td>
</tr>
<tr>
<td>static const Color</td>
<td>Yellow</td>
<td>Yellow predefined color. <a href="#">More...</a></td>
</tr>
<tr>
<td>static const Color</td>
<td>Magenta</td>
<td>Magenta predefined color. <a href="#">More...</a></td>
</tr>
<tr>
<td>static const Color</td>
<td>Cyan</td>
<td>Cyan predefined color. <a href="#">More...</a></td>
</tr>
<tr>
<td>static const Color</td>
<td>Transparent</td>
<td>Transparent (black) predefined color. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
### Related Functions

(Note that these are not member functions.)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool operator== (const Color &amp;left, const Color &amp;right)</td>
<td>Overload of the == operator.</td>
<td>More...</td>
</tr>
<tr>
<td>bool operator!= (const Color &amp;left, const Color &amp;right)</td>
<td>Overload of the != operator.</td>
<td>More...</td>
</tr>
<tr>
<td>Color operator+ (const Color &amp;left, const Color &amp;right)</td>
<td>Overload of the binary + operator.</td>
<td>More...</td>
</tr>
<tr>
<td>Color operator- (const Color &amp;left, const Color &amp;right)</td>
<td>Overload of the binary - operator.</td>
<td>More...</td>
</tr>
<tr>
<td>Color operator* (const Color &amp;left, const Color &amp;right)</td>
<td>Overload of the binary * operator.</td>
<td>More...</td>
</tr>
<tr>
<td>Color &amp; operator+= (Color &amp;left, const Color &amp;right)</td>
<td>Overload of the binary += operator.</td>
<td>More...</td>
</tr>
<tr>
<td>Color &amp; operator-= (Color &amp;left, const Color &amp;right)</td>
<td>Overload of the binary -= operator.</td>
<td>More...</td>
</tr>
<tr>
<td>Color &amp; operator*=(Color &amp;left, const Color &amp;right)</td>
<td>Overload of the binary *= operator.</td>
<td>More...</td>
</tr>
</tbody>
</table>
Utility class for manipulating RGBA colors.

sf::Color is a simple color class composed of 4 components:

- Red
- Green
- Blue
- Alpha (opacity)

Each component is a public member, an unsigned integer in the range [0, 255]. Thus, colors can be constructed and manipulated very easily:

```cpp
sf::Color color(255, 0, 0); // red
color.r = 0; // make it black
color.b = 128; // make it dark blue
```

The fourth component of colors, named "alpha", represents the opacity of the color. A value of 255 will be fully opaque, while an alpha value of 0 will make a color fully transparent, whatever the value of the other components is.

The most common colors are already defined as static variables:

```cpp
sf::Color black = sf::Color::Black;
sf::Color white = sf::Color::White;
sf::Color red = sf::Color::Red;
sf::Color green = sf::Color::Green;
sf::Color blue = sf::Color::Blue;
sf::Color yellow = sf::Color::Yellow;
sf::Color magenta = sf::Color::Magenta;
sf::Color cyan = sf::Color::Cyan;
sf::Color transparent = sf::Color::Transparent;
```
Colors can also be added and modulated (multiplied) using the overloaded operators + and *.
Definition at line 40 of file Color.hpp.
Constructor & Destructor Documentation

sf::Color::Color ( )

Default constructor.

Constructs an opaque black color. It is equivalent to sf::Color(0, 0, 0, 255)

sf::Color::Color ( Uint8 red,
                 Uint8 green,
                 Uint8 blue,
                 Uint8 alpha = 255 )

Construct the color from its 4 RGBA components.

Parameters
  red   Red component (in the range [0, 255])
  green Green component (in the range [0, 255])
  blue  Blue component (in the range [0, 255])
  alpha Alpha (opacity) component (in the range [0, 255])

sf::Color::Color ( Uint32 color )

Construct the color from 32-bit unsigned integer.
Parameters

**color** Number containing the RGBA components (in that order)
Member Function Documentation

**Uint32 sf::Color::toInteger ( ) const**

Retrieve the color as a 32-bit unsigned integer.

**Returns**

- Color represented as a 32-bit unsigned integer
bool operator!=( const Color & left, const Color & right )

Overload of the != operator.

This operator compares two colors and check if they are different.

**Parameters**
- *left* Left operand
- *right* Right operand

**Returns**
True if colors are different, false if they are equal

Color operator*( const Color & left, const Color & right )

Overload of the binary * operator.

This operator returns the component-wise multiplication (also called modulation). Components are then divided by 255 so that the result is still in the range [0, 255].

**Parameters**
- *left* Left operand
right Right operand

Returns
Result of $left \times right$

```cpp
Color & operator*=( Color & left,
                 const Color & right );
```

Overload of the binary $\times=$ operator.

This operator returns the component-wise multiplication (also called "modulation") assigns the result to the left operand. Components are then divided by 2 so that the result is still in the range $[0, 255]$. Parameters

- left Left operand
- right Right operand

Returns
Reference to $left$

```cpp
Color operator+ ( const Color & left,
                 const Color & right );
```

Overload of the binary + operator.

This operator returns the component-wise sum of two colors. Components to 255.
Parameters

left  Left operand
right Right operand

Returns

Result of \textit{left} + \textit{right}

\begin{verbatim}
Color & operator+= ( Color & left,
               const Color & right
                     )
\end{verbatim}

Overload of the binary += operator.

This operator computes the component-wise sum of two colors, and assigns the result to the left operand. Components that exceed 255 are clamped to 255.

Parameters

left  Left operand
right Right operand

Returns

Reference to \textit{left}

\begin{verbatim}
Color operator- ( const Color & left,
               const Color & right
                     )
\end{verbatim}

Overload of the binary - operator.

This operator returns the component-wise subtraction of two colors. Components below 0 are clamped to 0.
Parameters
    left  Left operand
    right Right operand

Returns
    Result of left - right

Color & operator-= ( Color & left,
                  const Color & right
                  )

Overload of the binary -= operator.

This operator computes the component-wise subtraction of two colors, the left operand. Components below 0 are clamped to 0.

Parameters
    left  Left operand
    right Right operand

Returns
    Reference to left

bool operator==( const Color & left,
                 const Color & right
                 )

Overload of the == operator.

This operator compares two colors and check if they are equal.
Parameters

- **left**  Left operand
- **right** Right operand

Returns

True if colors are equal, false if they are different
## Member Data Documentation

<table>
<thead>
<tr>
<th>Uint8 sf::Color::a</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha (opacity) component.</td>
<td></td>
</tr>
<tr>
<td>Definition at line 99 of file Color.hpp.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Uint8 sf::Color::b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue component.</td>
<td></td>
</tr>
<tr>
<td>Definition at line 98 of file Color.hpp.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>const Color sf::Color::Black</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Black predefined color.</td>
<td></td>
</tr>
<tr>
<td>Definition at line 83 of file Color.hpp.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>const Color sf::Color::Blue</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue predefined color.</td>
<td></td>
</tr>
<tr>
<td>Definition at line 87 of file Color.hpp.</td>
<td></td>
</tr>
<tr>
<td>Const</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Cyan</strong></td>
<td>Cyan predefined color.</td>
</tr>
<tr>
<td><strong>Green</strong></td>
<td>Green component.</td>
</tr>
<tr>
<td><strong>Magenta</strong></td>
<td>Magenta predefined color.</td>
</tr>
<tr>
<td><strong>r</strong></td>
<td>Uint8</td>
</tr>
<tr>
<td><strong>g</strong></td>
<td>Uint8</td>
</tr>
</tbody>
</table>
Red component.
Definition at line 96 of file Color.hpp.

\textbf{const Color sf::Color::Red}

Red predefined color.
Definition at line 85 of file Color.hpp.

\textbf{const Color sf::Color::Transparent}

Transparent (black) predefined color.
Definition at line 91 of file Color.hpp.

\textbf{const Color sf::Color::White}

White predefined color.
Definition at line 84 of file Color.hpp.

\textbf{const Color sf::Color::Yellow}

Yellow predefined color.
Definition at line 88 of file Color.hpp.
The documentation for this class was generated from the following file:
- Color.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::ConvexShape Class Reference
Graphics module

Specialized shape representing a convex polygon. More...

#include <ConvexShape.hpp>

Inheritance diagram for sf::ConvexShape:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConvexShape (std::size_t pointCount=0)</code></td>
<td>Default constructor.</td>
<td></td>
</tr>
<tr>
<td><code>void setPointCount (std::size_t count)</code></td>
<td>Set the number of points of the polygon.</td>
<td></td>
</tr>
<tr>
<td><code>virtual std::size_t getPointCount () const</code></td>
<td>Get the number of points of the polygon.</td>
<td></td>
</tr>
<tr>
<td><code>void setPoint (std::size_t index, const Vector2f &amp;point)</code></td>
<td>Set the position of a point.</td>
<td></td>
</tr>
<tr>
<td><code>virtual Vector2f getPoint (std::size_t index) const</code></td>
<td>Get the position of a point.</td>
<td></td>
</tr>
<tr>
<td><code>void setTexture (const Texture *texture, bool resetRect=false)</code></td>
<td>Change the source texture of the shape.</td>
<td></td>
</tr>
<tr>
<td><code>void setTextureRect (const IntRect &amp;rect)</code></td>
<td>Set the sub-rectangle of the texture that the shape will display.</td>
<td></td>
</tr>
<tr>
<td><code>void setFillColor (const Color &amp;color)</code></td>
<td>Set the fill color of the shape.</td>
<td></td>
</tr>
<tr>
<td><code>void setOutlineColor (const Color &amp;color)</code></td>
<td>Set the outline color of the shape.</td>
<td></td>
</tr>
<tr>
<td><code>void setOutlineThickness (float thickness)</code></td>
<td>Set the thickness of the shape's outline.</td>
<td></td>
</tr>
<tr>
<td><code>const Texture * getTexture () const</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><code>getTextureRect()</code></td>
<td>Get the source texture of the shape.</td>
<td></td>
</tr>
<tr>
<td><code>getFillColor()</code></td>
<td>Get the fill color of the shape.</td>
<td></td>
</tr>
<tr>
<td><code>getOutlineColor()</code></td>
<td>Get the outline color of the shape.</td>
<td></td>
</tr>
<tr>
<td><code>getOutlineThickness()</code></td>
<td>Get the outline thickness of the shape.</td>
<td></td>
</tr>
<tr>
<td><code>getLocalBounds()</code></td>
<td>Get the local bounding rectangle of the entity.</td>
<td></td>
</tr>
<tr>
<td><code>getGlobalBounds()</code></td>
<td>Get the global (non-minimal) bounding rectangle of the entity.</td>
<td></td>
</tr>
<tr>
<td><code>setPosition(float x, float y)</code></td>
<td>Set the position of the object.</td>
<td></td>
</tr>
<tr>
<td><code>setPosition(const Vector2f &amp;position)</code></td>
<td>Set the position of the object.</td>
<td></td>
</tr>
<tr>
<td><code>setRotation(float angle)</code></td>
<td>Set the orientation of the object.</td>
<td></td>
</tr>
<tr>
<td><code>setScale(float factorX, float factorY)</code></td>
<td>Set the scale factors of the object.</td>
<td></td>
</tr>
<tr>
<td><code>setScale(const Vector2f &amp;factors)</code></td>
<td>Set the scale factors of the object.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>void setOrigin(float x, float y)</code></td>
<td>set the local origin of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>void setOrigin(const Vector2f &amp;origin)</code></td>
<td>set the local origin of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>const Vector2f &amp; getPosition()</code></td>
<td>get the position of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>float getRotation()</code></td>
<td>get the orientation of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>const Vector2f &amp; getScale()</code></td>
<td>get the current scale of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>const Vector2f &amp; getOrigin()</code></td>
<td>get the local origin of the object <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>void move(float offsetX, float offsetY)</code></td>
<td>Move the object by a given offset. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>void move(const Vector2f &amp;offset)</code></td>
<td>Move the object by a given offset. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>void rotate(float angle)</code></td>
<td>Rotate the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>void scale(float factorX, float factorY)</code></td>
<td>Scale the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>void scale(const Vector2f &amp;factor)</code></td>
<td>Scale the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><code>const Transform &amp; getTransform()</code></td>
<td>get the combined transform of the object <a href="#">More...</a></td>
<td></td>
</tr>
</tbody>
</table>
const Transform & getInverseTransform () const
get the inverse of the combined transform of the obje
Protected Member Functions

```cpp
void update ()
    Recompute the internal geometry of the shape. More...
```
Detailed Description

Specialized shape representing a convex polygon.

This class inherits all the functions of `sf::Transformable` (position, rotation functions of `sf::Shape` (outline, color, texture, ...).

It is important to keep in mind that a convex shape must always be drawn correctly. Moreover, the points must be defined in order; using a incorrect shape.

Usage example:

```cpp
sf::ConvexShape polygon;
polygon.setPointCount(3);
polygon.setPoint(0, sf::Vector2f(0, 0));
polygon.setPoint(1, sf::Vector2f(0, 10));
polygon.setPoint(2, sf::Vector2f(25, 5));
polygon.setOutlineColor(sf::Color::Red);
polygon.setOutlineThickness(5);
polygon.setPosition(10, 20);
...
window.draw(polygon);
```

See also

- `sf::Shape`, `sf::RectangleShape`, `sf::CircleShape`

Definition at line 42 of file `ConvexShape.hpp`. 
sf::ConvexShape::ConvexShape ( std::size_t pointCount = 0 )

Default constructor.

**Parameters**

- `pointCount` Number of points of the polygon
Member Function Documentation

**const Color**& sf::Shape::getFillColor () const

Get the fill color of the shape.

**Returns**
Fill color of the shape

**See also**
setFillColor

**FloatRect** sf::Shape::getGlobalBounds () const

Get the global (non-minimal) bounding rectangle of the entity.

The returned rectangle is in global coordinates, which means that all transformations (translation, rotation, scale, ...) that are applied to the entity are taken into account. The returned rectangle is the bounds of the shape in the global 2D world's coordinate system.

This function does not necessarily return the minimal bounding rectangle. It merely ensures that the returned rectangle covers all the vertices (but possibly more). This allows for a fast approximation as a first check; you may want to use more precise checks on top.

**Returns**
Global bounding rectangle of the entity
**const Transform** & sf::Transformable::getInverseTransform() const

get the inverse of the combined transform of the object

**Returns**
Inverse of the combined transformations applied to the object

**See also**
getTransform

**FloatRect** sf::Shape::getLocalBounds() const

Get the local bounding rectangle of the entity.

The returned rectangle is in local coordinates, which means that (translation, rotation, scale, ...) that are applied to the entity. In other bounds of the entity in the entity's coordinate system.

**Returns**
Local bounding rectangle of the entity

**const Vector2f** & sf::Transformable::getOrigin() const

get the local origin of the object

**Returns**
Current origin

**See also**
setOrigin
**const Color& sf::Shape::getOutlineColor ( ) const**

Get the outline color of the shape.

**Returns**

Outline color of the shape

**See also**

setOutlineColor

**float sf::Shape::getOutlineThickness ( ) const**

Get the outline thickness of the shape.

**Returns**

Outline thickness of the shape

**See also**

setOutlineThickness

**virtual Vector2f sf::ConvexShape::getPoint ( std::size_t index ) const**

Get the position of a point.

The returned point is in local coordinates, that is, the shape's transforms taken into account. The result is undefined if index is out of the valid range.

**Parameters**

*index* Index of the point to get, in range [0 .. getPointCount() - 1]
Returns
Position of the index-th point of the polygon

See also
setPoint

Implements sf::Shape.

virtual std::size_t sf::ConvexShape::getPointCount () const

Get the number of points of the polygon.

Returns
Number of points of the polygon

See also
setPointCount

Implements sf::Shape.

const Vector2f& sf::Transformable::getPosition () const

get the position of the object

Returns
Current position

See also
setPosition
**float sf::Transformable::getRotation ( ) const**

get the orientation of the object

The rotation is always in the range [0, 360].

**Returns**

Current rotation, in degrees

**See also**

setRotation

**const Vector2f& sf::Transformable::getScale ( ) const**

get the current scale of the object

**Returns**

Current scale factors

**See also**

setScale

**const Texture* sf::Shape::getTexture ( ) const**

Get the source texture of the shape.

If the shape has no source texture, a NULL pointer is returned. This means that you can't modify the texture when you retrieve it with this function.

**Returns**

Pointer to the shape's texture
**See also**

`setTexture`  

**const IntRect** & `sf::Shape::getTextureRect ( ) const`

Get the sub-rectangle of the texture displayed by the shape.

**Returns**

- `Texture` rectangle of the shape

**See also**

`setTextureRect`

**const Transform** & `sf::Transformable::getTransform ( ) const`

get the combined transform of the object

**Returns**

- `Transform` combining the position/rotation/scale/origin of the object

**See also**

`getInverseTransform`

**void sf::Transformable::move ( float offsetX, float offsetY )**

Move the object by a given offset.
This function adds to the current position of the object, unlike setPosition which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f pos = object.getPosition();
object.setPosition(pos.x + offsetX, pos.y + offsetY);
```

**Parameters**
- **offsetX** X offset
- **offsetY** Y offset

**See also**
- setPosition

---

```cpp
void sf::Transformable::move ( const Vector2f & offset )
```

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setPosition(object.getPosition() + offset);
```

**Parameters**
- **offset** Offset

**See also**
- setPosition

---

```cpp
void sf::Transformable::rotate ( float angle )
```

Rotate the object.
This function adds to the current rotation of the object, unlike setRotation which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setRotation(object.getRotation() + angle);
```

**Parameters**

- **angle** Angle of rotation, in degrees

---

```cpp
void sf::Transformable::scale (float factorX, float factorY)
```

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factorX, scale.y * factorY);
```

**Parameters**

- **factorX** Horizontal scale factor
- **factorY** Vertical scale factor

**See also**

- `setScale`

---

```cpp
void sf::Transformable::scale (const Vector2f & factor)
```

Scale the object.
This function multiplies the current scale of the object, unlike `setScale` which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factor.x, scale.y * factor.y);
```

**Parameters**

- `factor` Scale factors

**See also**
- `setScale`

---

**void sf::Shape::setFillColor (const Color & color)**

Set the fill color of the shape.

This color is modulated (multiplied) with the shape's texture if any. It can be used to colorize the shape, or change its global opacity. You can use `sf::Color::Transparent` to make the inside of the shape transparent, and have the outline alone. By default, the shape's fill color is opaque white.

**Parameters**

- `color` New color of the shape

**See also**
- `getFillColor`, `setOutlineColor`

---

**void sf::Transformable::setOrigin (float x, float y)**

set the local origin of the object
The origin of an object defines the center point for all transformations. Coordinates of this point must be relative to the top-left corner transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

**Parameters**

- **x** X coordinate of the new origin
- **y** Y coordinate of the new origin

**See also**

`getOrigin`

---

```cpp
void sf::Transformable::setOrigin ( const Vector2f & origin )
```

set the local origin of the object

The origin of an object defines the center point for all transformations. Coordinates of this point must be relative to the top-left corner transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

**Parameters**

- **origin** New origin

**See also**

`getOrigin`

---

```cpp
void sf::Shape::setOutlineColor ( const Color & color )
```

Set the outline color of the shape.

By default, the shape's outline color is opaque white.
**Parameters**

- **color** New outline color of the shape

**See also**

- `getOutlineColor`, `setFillcolor`

---

```cpp
void sf::Shape::setOutlineThickness ( float thickness )
```

Set the thickness of the shape's outline.

Note that negative values are allowed (so that the outline expands towards the center of the shape), and using zero disables the outline. By default, the outline thickness is 0.

**Parameters**

- **thickness** New outline thickness

**See also**

- `getOutlineThickness`

---

```cpp
void sf::ConvexShape::setPoint ( std::size_t index, const Vector2f & point )
```

Set the position of a point.

Don't forget that the polygon must remain convex, and the points need to be ordered! `setPointCount` must be called first in order to set the total number of points. The result is undefined if `index` is outside the valid range.

**Parameters**

- **index** Index of the point to change, in range `[0 .. getPointCount() - 1]`
**point**  New position of the point

See also

  * getPoint

---

**void sf::ConvexShape::setPointCount ( std::size_t count )**

Set the number of points of the polygon.

*count* must be greater than 2 to define a valid shape.

**Parameters**

  * count New number of points of the polygon

See also

  * getPointCount

---

**void sf::Transformable::setPosition ( float x, float y )**

Set the position of the object

This function completely overwrites the previous position. See the move function to apply an offset based on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**

  * x X coordinate of the new position
  * y Y coordinate of the new position

See also
```cpp
void sf::Transformable::setPosition ( const Vector2f & position )
```

set the position of the object

This function completely overwrites the previous position. See the move function on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**

- **position** New position

**See also**

- move, getPosition

```cpp
void sf::Transformable::setRotation ( float angle )
```

set the orientation of the object

This function completely overwrites the previous rotation. See the rotate function on the previous rotation instead. The default rotation of a transformable object is 0.

**Parameters**

- **angle** New rotation, in degrees

**See also**

- rotate, getRotation

```cpp
void sf::Transformable::setScale ( float factorX, float factorY )
```
set the scale factors of the object

This function completely overwrites the previous scale. See the scale function instead. The default scale of a transformable object is (1, 1).

**Parameters**

- **factorX** New horizontal scale factor
- **factorY** New vertical scale factor

**See also**

- `scale`, `getScale`

---

```cpp
void sf::Transformable::setScale ( const Vector2f & factors )
```

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function instead. The default scale of a transformable object is

**Parameters**

- **factors** New scale factors

**See also**

- `scale`, `getScale`

---

```cpp
void sf::Shape::setTexture ( const Texture * texture, bool resetRect = false )
```
Change the source texture of the shape.

The `texture` argument refers to a texture that must exist as long as the shape exists. The shape doesn't store its own copy of the texture, but rather keeps a pointer to it. If the source texture is destroyed and the shape tries to use it, the behavior is undefined. The `texture` can be NULL to disable texturing. If `resetRect` is true, the TextureRect property is automatically adjusted to the size of the new texture. If it is false, the texture rect is left unchanged.

**Parameters**

- `texture` New texture
- `resetRect` Should the texture rect be reset to the size of the new texture?

**See also**

- `getTexture`, `setTextureRect`

---

```cpp
void sf::Shape::setTextureRect (const IntRect & rect )
```

Set the sub-rectangle of the texture that the shape will display.

The texture rect is useful when you don't want to display the whole texture. By default, the texture rect covers the entire texture.

**Parameters**

- `rect` Rectangle defining the region of the texture to display

**See also**

- `getTextureRect`, `setTexture`

---

```cpp
void sf::Shape::update ( )
```
Recompute the internal geometry of the shape.

This function must be called by the derived class everytime the shape's either getPointCount or getPoint is different.

The documentation for this class was generated from the following file:
- ConvexShape.hpp
sf::Drawable Class Reference

Abstract base class for objects that can be drawn to a render target. More

#include <Drawable.hpp>

Inheritance diagram for sf::Drawable:
Public Member Functions

```
virtual ~Drawable ()
Virtual destructor. More...
```
**Protected Member Functions**

<table>
<thead>
<tr>
<th>virtual void    draw (RenderTarget &amp;target, RenderStates states) const =0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw the object to a render target. More...</td>
</tr>
</tbody>
</table>
# Friends

| class | RenderTarget |
Abstract base class for objects that can be drawn to a render target.

`sf::Drawable` is a very simple base class that allows objects of derived classes to be drawn to a `sf::RenderTarget`.

All you have to do in your derived class is to override the `draw` virtual function.

Note that inheriting from `sf::Drawable` is not mandatory, but it allows this syntax "window.draw(object)" rather than "object.draw(window)", which is more consistent with other SFML classes.

Example:

```cpp
class MyDrawable : public sf::Drawable
{
public:
    ...

private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
        // You can draw other high-level objects
        target.draw(m_sprite, states);

        // ... or use the low-level API
        states.texture = &m_texture;
        target.draw(m_vertices, states);

        // ... or draw with OpenGL directly
        glBegin(GL_QUADS);
        ...
        glEnd();
    }

    sf::Sprite m_sprite;
    sf::Texture m_texture;
    sf::VertexArray m_vertices;
};
```
See also
sf::RenderTarget

Definition at line 44 of file Drawable.hpp.
Virtual destructor.

Definition at line 52 of file Drawable.hpp.
virtual void sf::Drawable::draw (RenderTarget & target, RenderStates states const)

Draw the object to a render target.

This is a pure virtual function that has to be implemented by the derived classes to define how the drawable should be drawn.

**Parameters**
- **target** Render target to draw to
- **states** Current render states

The documentation for this class was generated from the following file:
- **Drawable.hpp**
sf::Font Class Reference

Graphics module

Class for loading and manipulating character fonts. More...

#include <Font.hpp>
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Holds various information about a font. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Font ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Font (const Font &amp;copy)</strong></td>
<td>Copy constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>~Font ()</strong></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool loadFromFile (const std::string &amp;filename)</strong></td>
<td>Load the font from a file. <a href="#">More...</a></td>
</tr>
<tr>
<td>*<em>bool loadFromMemory (const void <em>data, std::size_t sizeInBytes)</em></em></td>
<td>Load the font from a file in memory. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool loadFromStream (InputStream &amp;stream)</strong></td>
<td>Load the font from a custom stream. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const Info &amp; getInfo () const</strong></td>
<td>Get the font information. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const Glyph &amp; getGlyph (Uint32 codePoint, unsigned int characterSize, float outlineThickness=0) const</strong></td>
<td>Retrieve a glyph of the font. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>float getKerning (Uint32 first, Uint32 second, unsigned int characterSize)</strong></td>
<td>Get the kerning offset of two glyphs. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>float getLineSpacing (unsigned int characterSize) const</strong></td>
<td>Get the line spacing. <a href="#">More...</a></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>float getUnderlinePosition (unsigned int characterSize) const</code></td>
<td>Get the position of the underline. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>float getUnderlineThickness (unsigned int characterSize) const</code></td>
<td>Get the thickness of the underline. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>const Texture &amp; getTexture (unsigned int characterSize) const</code></td>
<td>Retrieve the texture containing the loaded glyphs of a certain size.</td>
</tr>
<tr>
<td><code>Font &amp; operator= (const Font &amp;right)</code></td>
<td>Overload of assignment operator. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Class for loading and manipulating character fonts.

Fonts can be loaded from a file, from memory or from a custom stream, types of fonts.

See the loadFromFile function for the complete list of supported formats.

Once it is loaded, a `sf::Font` instance provides three types of information about fonts:

- Global metrics, such as the line spacing
- Per-glyph metrics, such as bounding box or kerning
- Pixel representation of glyphs

Fonts alone are not very useful: they hold the font data but cannot make anything useful out of it. To do so you need to use the `sf::Text` class, which is able to properly output text with several options such as character size, style, color, position, rotation, etc. This separation allows more flexibility: indeed a `sf::Font` is a heavy resource, and any operation on it is slow (often too slow for real-time applications). On the other side, a `sf::Text` is a lightweight object which can combine the metrics of a `sf::Font` to display any text on a render target. Note that it is possible to bind several `sf::Text` instances to the same `sf::Font`.

It is important to note that the `sf::Text` instance doesn't copy the font that it uses to it. Thus, a `sf::Font` must not be destructed while it is used by a `sf::Text` (which uses a local `sf::Font` instance for creating a text).

Usage example:

```cpp
// Declare a new font
sf::Font font;
```
// Load it from a file
if (!font.loadFromFile("arial.ttf"))
{
    // error...
}

// Create a text which uses our font
sf::Text text1;
text1.setFont(font);
text1.setCharacterSize(30);
text1.setStyle(sf::Text::Regular);

// Create another text using the same font, but with different parameters
sf::Text text2;
text2.setFont(font);
text2.setCharacterSize(50);
text2.setStyle(sf::Text::Italic);

Apart from loading font files, and passing them to instances of sf::Text, deal directly with this class. However, it may be useful to access the font for advanced usage.

Note that if the font is a bitmap font, it is not scalable, thus not all requested sizes will be available to use. This needs to be taken into consideration when using sf::Text. If you need to display text of a certain size, make sure the corresponding bitmap font that supports that size is used.

See also

sf::Text

Definition at line 50 of file Font.hpp.
Constructor & Destructor Documentation

sf::Font::Font ( )

Default constructor.

This constructor defines an empty font

sf::Font::Font ( const Font & copy )

Copy constructor.

Parameters

- **copy** Instance to copy

sf::Font::~Font ( )

Destructor.

Cleans up all the internal resources used by the font
Member Function Documentation

### `const Glyph& sf::Font::getGlyph ( Uint32 codePoint, unsigned int characterSize, bool bold, float outlineThickness = 0 ) const`

Retrieve a glyph of the font.

If the font is a bitmap font, not all character sizes might be available. If requested size, an empty glyph is returned.

Be aware that using a negative value for the outline thickness will cause

**Parameters**

- **codePoint**
  - Unicode code point of the character to get
- **characterSize**
  - Reference character size
- **bold**
  - Retrieve the bold version or the regular one?
- **outlineThickness**
  - Thickness of outline (when != 0 the glyph will not

**Returns**

The glyph corresponding to `codePoint` and `characterSize`

### `const Info& sf::Font::getInfo ( ) const`

Get the font information.
**Returns**

A structure that holds the font information

```
float sf::Font::getKerning ( Uint32 first,
    Uint32 second,
    unsigned int characterSize
) const
```

Get the kerning offset of two glyphs.

The kerning is an extra offset (negative) to apply between two glyphs within a pair look more "natural". For example, the pair "AV" have a special kerning to make them closer than other characters. Most of the glyphs pairs have a kerning offset of zero, though.

**Parameters**

- `first` Unicode code point of the first character
- `second` Unicode code point of the second character
- `characterSize` Reference character size

**Returns**

Kerning value for `first` and `second`, in pixels

```
float sf::Font::getLineSpacing ( unsigned int characterSize ) const
```

Get the line spacing.

Line spacing is the vertical offset to apply between two consecutive lines.

**Parameters**

- `characterSize` Reference character size
Returns

Line spacing, in pixels

const Texture& sf::Font::getTexture ( unsigned int characterSize ) const

Retrieve the texture containing the loaded glyphs of a certain size.

The contents of the returned texture changes as more glyphs are requested. It is mainly used internally by sf::Text.

Parameters

characterSize Reference character size

Returns

Texture containing the glyphs of the requested size

float sf::Font::getUnderlinePosition ( unsigned int characterSize ) const

Get the position of the underline.

Underline position is the vertical offset to apply between the baseline and the underline.

Parameters

characterSize Reference character size

Returns

Underline position, in pixels

See also

getUnderlineThickness
**float sf::Font::getUnderlineThickness ( unsigned int characterSize )**

Get the thickness of the underline.

Underline thickness is the vertical size of the underline.

**Parameters**

- **characterSize** Reference character size

**Returns**

- Underline thickness, in pixels

**See also**

- getUnderlinePosition

---

**bool sf::Font::loadFromFile ( const std::string & filename )**

Load the font from a file.

The supported font formats are: TrueType, Type 1, CFF, OpenType, \(^*\) BDF, PFR and Type 42. Note that this function know nothing about the user's system, thus you can't load them directly.

**Warning**

- SFML cannot preload all the font data in this function, so the file has to remain accessible until the **sf::Font** object loads a new font or is destroyed.

**Parameters**

- **filename** Path of the font file to load

**Returns**
True if loading succeeded, false if it failed

See also
loadFromMemory, loadFromStream

```cpp
bool sf::Font::loadFromMemory ( const void * data,
                        std::size_t sizeInBytes )
```

Load the font from a file in memory.

The supported font formats are: TrueType, Type 1, CFF, OpenType, \* BDF, PFR and Type 42.

**Warning**

SFML cannot preload all the font data in this function, so the buffer is valid until the sf::Font object loads a new font or is destroyed.

**Parameters**

- `data` Pointer to the file data in memory
- `sizeInBytes` Size of the data to load, in bytes

**Returns**

True if loading succeeded, false if it failed

See also
loadFromFile, loadFromStream

```cpp
bool sf::Font::loadFromStream ( InputStream & stream )
```

Load the font from a custom stream.
The supported font formats are: TrueType, Type 1, CFF, OpenType, BDF, PFR and Type 42. Warning: SFML cannot preload all the font data of stream have to remain valid as long as the font is used.

**Warning**

SFML cannot preload all the font data in this function, so the stream the sf::Font object loads a new font or is destroyed.

**Parameters**

*stream* Source stream to read from

**Returns**

True if loading succeeded, false if it failed

**See also**

loadFromFile, loadFromMemory

---

**Font& sf::Font::operator=(const Font & right)**

Overload of assignment operator.

**Parameters**

*right* Instance to assign

**Returns**

Reference to self

---

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

- Font.hpp

---

SFML is licensed under the terms and conditions of the zlib/png license
sf::Font::Info Struct Reference

Holds various information about a font. More...

#include <Font.hpp>
Public Attributes

<table>
<thead>
<tr>
<th>std::string family</th>
<th>The font family.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>More...</td>
</tr>
</tbody>
</table>
Detailed Description

Holds various information about a font.

Definition at line 58 of file Font.hpp.
std::string sf::Font::Info::family

The font family.

Definition at line 60 of file Font.hpp.

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

- Font.hpp
sf::Glyph Class Reference
Graphics module

Structure describing a glyph. More...

#include <Glyph.hpp>
Public Member Functions

**Glyph ()**
Default constructor. More...
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td><code>advance</code></td>
<td>Offset to move horizontally to the next character.</td>
</tr>
<tr>
<td>FloatRect</td>
<td><code>bounds</code></td>
<td>Bounding rectangle of the glyph, in coordinates relative to the</td>
</tr>
<tr>
<td>IntRect</td>
<td><code>textureRect</code></td>
<td>Texture coordinates of the glyph inside the font's texture.</td>
</tr>
</tbody>
</table>
Detailed Description

Structure describing a glyph.

A glyph is the visual representation of a character.

The sf::Glyph structure provides the information needed to handle the glyph:

- its coordinates in the font's texture
- its bounding rectangle
- the offset to apply to get the starting position of the next glyph

See also

  sf::Font

Definition at line 41 of file Glyph.hpp.
sf::Glyph::Glyph ()

Default constructor.

Definition at line 49 of file Glyph.hpp.
**Member Data Documentation**

<table>
<thead>
<tr>
<th><strong>float sf::Glyph::advance</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset to move horizontally to the next character.</td>
</tr>
<tr>
<td>Definition at line 54 of file Glyph.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>FloatRect sf::Glyph::bounds</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounding rectangle of the glyph, in coordinates relative to the baseline.</td>
</tr>
<tr>
<td>Definition at line 55 of file Glyph.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IntRect sf::Glyph::textureRect</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture coordinates of the glyph inside the font's texture.</td>
</tr>
<tr>
<td>Definition at line 56 of file Glyph.hpp.</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following file:
- Glyph.hpp
sf::Image Class Reference
Graphics module

Class for loading, manipulating and saving images. More...

#include <Image.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Image()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>~Image()</code></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void create(unsigned int width, unsigned int height, const Color&amp; color)</code></td>
<td>Create the image and fill it with a unique color. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void create(unsigned int width, unsigned int height, const Uint8*pixels)</code></td>
<td>Create the image from an array of pixels. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool loadFromFile(const std::string &amp;filename)</code></td>
<td>Load the image from a file on disk. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool loadFromMemory(const void *data, std::size_t size)</code></td>
<td>Load the image from a file in memory. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool loadFromStream(InputStream &amp;stream)</code></td>
<td>Load the image from a custom stream. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool saveToFile(const std::string &amp;filename)</code></td>
<td>Save the image to a file on disk. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>Vector2u getSize()</code></td>
<td>Return the size (width and height) of the image. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void createMaskFromColor(const Color&amp; color, Uint8 alpha=0)</code></td>
<td>Create a transparency mask from a specified color-key. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void copy(const Image &amp;source, unsigned int destX, unsigned int destY)</code></td>
<td>Copy the image from another. <a href="#">More...</a></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>void &amp;sourceRect=IntRect(0, 0, 0, 0), bool applyAlpha=false</td>
<td>Copy pixels from another image onto this one.</td>
</tr>
<tr>
<td>void setPixel (unsigned int x, unsigned int y, const Color &amp;color)</td>
<td>Change the color of a pixel.</td>
</tr>
<tr>
<td>Color getPixel (unsigned int x, unsigned int y) const</td>
<td>Get the color of a pixel.</td>
</tr>
<tr>
<td>const Uint8 * getPixelsPtr () const</td>
<td>Get a read-only pointer to the array of pixels.</td>
</tr>
<tr>
<td>void flipHorizontally ()</td>
<td>Flip the image horizontally (left &lt;-&gt; right)</td>
</tr>
<tr>
<td>void flipVertically ()</td>
<td>Flip the image vertically (top &lt;-&gt; bottom)</td>
</tr>
</tbody>
</table>
Detailed Description

Class for loading, manipulating and saving images.

`sfe::Image` is an abstraction to manipulate images as bidimensional arrays.

The class provides functions to load, read, write and save pixels, as well as many other useful functions.

`sfe::Image` can handle a unique internal representation of pixels, which is an RGBA 32 bits. This means that a pixel must be composed of 8 bits red, green, blue and alpha channels. Functions that return an array of pixels follow this rule, and all parameters that you pass to functions (such as `loadFromMemory`) must use this representation as well.

A `sf::Image` can be copied, but it is a heavy resource and if possible it is recommended to use references to pass or return them to avoid useless copies.

Usage example:

```cpp
// Load an image file from a file
sf::Image background;
if (!background.loadFromFile("background.jpg"))
    return -1;

// Create a 20x20 image filled with black color
sf::Image image;
image.create(20, 20, sf::Color::Black);

// Copy image1 on image2 at position (10, 10)
image.copy(background, 10, 10);

// Make the top-left pixel transparent
sf::Color color = image.getPixel(0, 0);
color.a = 0;
image.setPixel(0, 0, color);

// Save the image to a file
if (!image.saveToFile("result.png"))
    return -1;
```
See also

sf::Texture

Definition at line 46 of file Image.hpp.
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>sf::Image::Image ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default constructor.</td>
</tr>
<tr>
<td>Creates an empty image.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf::Image::~Image ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destructor.</td>
</tr>
</tbody>
</table>
Member Function Documentation

**void sf::Image::copy** ( const *Image* & *source*,
unsigned int *destX*,
unsigned int *destY*,
const *IntRect* & *sourceRect* = *IntRect*(0, 0, 0),
bool *applyAlpha* = false )

Copy pixels from another image onto this one.

This function does a slow pixel copy and should not be used intensively.
Complex static image from several others, but if you need this kind of feature,
sf::RenderTexture.

If *sourceRect* is empty, the whole image is copied. If *applyAlpha* is set to
true, pixels is applied. If it is false, the pixels are copied unchanged with their
alpha value.

**Parameters**

- **source**: Source image to copy
- **destX**: X coordinate of the destination position
- **destY**: Y coordinate of the destination position
- **sourceRect**: Sub-rectangle of the source image to copy
- **applyAlpha**: Should the copy take into account the source transparency?

**void sf::Image::create** ( unsigned int *width*,
unsigned int *height*,
const *Color* & *color* = *color*(0, 0, 0) )


Create the image and fill it with a unique color.

**Parameters**
- `width` Width of the image
- `height` Height of the image
- `color` Fill color

```cpp
void sf::Image::create(unsigned int width, unsigned int height, const Uint8 *pixels)
```

Create the image from an array of pixels.

The *pixel* array is assumed to contain 32-bits RGBA pixels, and have this is an undefined behavior. If `pixels` is null, an empty image is created.

**Parameters**
- `width` Width of the image
- `height` Height of the image
- `pixels` Array of pixels to copy to the image

```cpp
void sf::Image::createMaskFromColor(const Color &color, Uint8 alpha = 0)
```

Create a transparency mask from a specified color-key.
This function sets the alpha value of every pixel matching the given color to 0, making them become transparent.

**Parameters**

- **color**  
  Color to make transparent  
- **alpha**  
  Alpha value to assign to transparent pixels

```cpp
void sf::Image::flipHorizontally ()
```

Flip the image horizontally (left <-> right)

```cpp
void sf::Image::flipVertically ()
```

Flip the image vertically (top <-> bottom)

```cpp
Color sf::Image::getPixel (unsigned int x, unsigned int y, const )
```

Get the color of a pixel.

This function doesn't check the validity of the pixel coordinates, using out-of-range values will result in an undefined behavior.

**Parameters**

- **x**  
  X coordinate of pixel to get  
- **y**  
  Y coordinate of pixel to get

**Returns**
Color of the pixel at coordinates (x, y)

See also
setPixel

const Uint8* sf::Image::getPixelsPtr () const

Get a read-only pointer to the array of pixels.

The returned value points to an array of RGBA pixels made of 8 bits in the array is width * height * 4 (getSize().x * getSize().y * 4). Warning: the invalid if you modify the image, so you should never store it for too long. If the image is empty, a null pointer is returned.

Returns
Read-only pointer to the array of pixels

Vector2u sf::Image::getSize () const

Return the size (width and height) of the image.

Returns
Size of the image, in pixels

bool sf::Image::loadFromFile (const std::string & filename)

Load the image from a file on disk.

The supported image formats are bmp, png, tga, jpg, gif, psd, hdr and t.
supported, like progressive jpeg. If this function fails, the image is left unchanged.

**Parameters**

- **filename** Path of the image file to load

**Returns**

- True if loading was successful

**See also**

- `loadFromFile`, `loadFromStream`, `saveToFile`

---

```cpp
bool sf::Image::loadFromMemory ( const void * data,
                                  std::size_t size )
```

Load the image from a file in memory.

The supported image formats arebmp, png, tga, jpg, gif, psd, hdr and `pic`. Some format options are not supported, like progressive jpeg. If this function fails, the image is left unchanged.

**Parameters**

- **data** Pointer to the file data in memory
- **size** Size of the data to load, in bytes

**Returns**

- True if loading was successful

**See also**

- `loadFromFile`, `loadFromStream`

---

```cpp
bool sf::Image::loadFromStream ( InputStream & stream )
```
Load the image from a custom stream.

The supported image formats are bmp, png, tga, jpg, gif, psd, hdr and ps, supported, like progressive jpeg. If this function fails, the image is left unchanged.

**Parameters**
- `stream` Source stream to read from

**Returns**
- True if loading was successful

**See also**
- loadFromFile, loadFromMemory

```cpp
bool sf::Image::saveToFile (const std::string & filename) const
```

Save the image to a file on disk.

The format of the image is automatically deduced from the extension. The supported image formats are bmp, png, tga and jpg. The destination file is overwritten if it already exists.

**Parameters**
- `filename` Path of the file to save

**Returns**
- True if saving was successful

**See also**
- create, loadFromFile, loadFromMemory
void sf::Image::setPixel ( unsigned int x, 
unsigned int y, 
const Color & color )

Change the color of a pixel.

This function doesn't check the validity of the pixel coordinates, using out-of-range values will result in undefined behavior.

**Parameters**
- **x** X coordinate of pixel to change
- **y** Y coordinate of pixel to change
- **color** New color of the pixel

**See also**
- getPixel

The documentation for this class was generated from the following file:
- Image.hpp

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Rect< T > Class Template Reference

Graphics module

Utility class for manipulating 2D axis aligned rectangles. More...

#include <Rect.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rect ()</td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>Rect (T rectLeft, T rectTop, T rectWidth, T rectHeight)</td>
<td>Construct the rectangle from its coordinates. <a href="#">More...</a></td>
</tr>
<tr>
<td>Rect (const Vector2&lt;T&gt; &amp;position, const Vector2&lt;T&gt; &amp;size)</td>
<td>Construct the rectangle from position and size. <a href="#">More...</a></td>
</tr>
<tr>
<td>template&lt;typename U &gt; Rect (const Rect&lt;U&gt; &amp;rectangle)</td>
<td>Construct the rectangle from another type of rectangle. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool contains (T x, T y) const</td>
<td>Check if a point is inside the rectangle's area. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool contains (const Vector2&lt;T&gt; &amp;point) const</td>
<td>Check if a point is inside the rectangle's area. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool intersects (const Rect&lt;T&gt; &amp;rectangle) const</td>
<td>Check the intersection between two rectangles. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool intersects (const Rect&lt;T&gt; &amp;rectangle, Rect&lt;T&gt; &amp;intersection) const</td>
<td>Check the intersection between two rectangles. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Public Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Left coordinate of the rectangle. <a href="#">More...</a></td>
</tr>
<tr>
<td>top</td>
<td>Top coordinate of the rectangle. <a href="#">More...</a></td>
</tr>
<tr>
<td>width</td>
<td>Width of the rectangle. <a href="#">More...</a></td>
</tr>
<tr>
<td>height</td>
<td>Height of the rectangle. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Related Functions

(Note that these are not member functions.)

```cpp
template<typename T >
bool operator== (const Rect< T > &left, const Rect< T > &right)
     Overload of binary operator ==. More...
```

```cpp
template<typename T >
bool operator!= (const Rect< T > &left, const Rect< T > &right)
     Overload of binary operator !=. More...
```
Detailed Description

\texttt{template<typename T> class sf::Rect< T >}

Utility class for manipulating 2D axis aligned rectangles.

A rectangle is defined by its top-left corner and its size.

It is a very simple class defined for convenience, so its member variables are public and can be accessed directly, just like the vector classes (\texttt{Vector2} and \texttt{Vector3}).

To keep things simple, \texttt{sf::Rect} doesn't define functions to emulate the non-member members (such as right, bottom, center, etc.), it rather only provides intersection functions.

\texttt{sf::Rect} uses the usual rules for its boundaries:

- The left and top edges are included in the rectangle's area
- The right (left + width) and bottom (top + height) edges are excluded from the rectangle's area

This means that \texttt{sf::IntRect(0, 0, 1, 1)} and \texttt{sf::IntRect(1, 1, 1, 1)} don't intersect.

\texttt{sf::Rect} is a template and may be used with any numeric type, but for simplicity SFML are typedef'd:

- \texttt{sf::Rect<int>} is \texttt{sf::IntRect}
- \texttt{sf::Rect<float>} is \texttt{sf::FloatRect}

So that you don't have to care about the template syntax.

Usage example:
// Define a rectangle, located at (0, 0) with a size of 20x5
sf::IntRect r1(0, 0, 20, 5);

// Define another rectangle, located at (4, 2) with a size of 18x10
sf::Vector2i position(4, 2);
sf::Vector2i size(18, 10);
sf::IntRect r2(position, size);

// Test intersections with the point (3, 1)
bool b1 = r1.contains(3, 1); // true
bool b2 = r2.contains(3, 1); // false

// Test the intersection between r1 and r2
sf::IntRect result;
bool b3 = r1.intersects(r2, result); // true
// result == (4, 2, 16, 3)

Definition at line 42 of file Rect.hpp.
Constructor & Destructor Documentation

```cpp
template<typename T>
sf::Rect<T>::Rect();
```

Default constructor.

Creates an empty rectangle (it is equivalent to calling Rect(0, 0, 0, 0)).

```cpp
template<typename T>
sf::Rect<T>::Rect(T rectLeft, T rectTop, T rectWidth, T rectHeight);
```

Construct the rectangle from its coordinates.

Be careful, the last two parameters are the width and height, not the right and bottom coordinates!

**Parameters**
- `rectLeft`  Left coordinate of the rectangle
- `rectTop`  Top coordinate of the rectangle
- `rectWidth`  Width of the rectangle
- `rectHeight`  Height of the rectangle
template<typename T>
sf::Rect<T>::Rect (const Vector2<T> & position, const Vector2<T> & size)

Construct the rectangle from position and size.

Be careful, the last parameter is the size, not the bottom-right corner!

Parameters

- **position** Position of the top-left corner of the rectangle
- **size** Size of the rectangle

template<typename T>
template<typename U >
sf::Rect<T>::Rect (const Rect<U> & rectangle)

Construct the rectangle from another type of rectangle.

This constructor doesn't replace the copy constructor, it's called only when U != T. A call to this constructor will fail to compile if U is not convertible to T.

Parameters

- **rectangle** Rectangle to convert
Member Function Documentation

```cpp
template<typename T>
bool sf::Rect<T>::contains ( T x, T y ) const
```

Check if a point is inside the rectangle's area.

This check is non-inclusive. If the point lies on the edge of the rectangle,

**Parameters**
- `x` X coordinate of the point to test
- `y` Y coordinate of the point to test

**Returns**
- True if the point is inside, false otherwise

**See also**
- `intersects`

```cpp
template<typename T>
bool sf::Rect<T>::contains ( const Vector2<T> & point ) const
```

Check if a point is inside the rectangle's area.

This check is non-inclusive. If the point lies on the edge of the rectangle,
Parameters
  point Point to test

Returns
  True if the point is inside, false otherwise

See also
  intersects

---

template<typename T>

bool sf::Rect<T>::intersects ( const Rect<T> & rectangle ) const

Check the intersection between two rectangles.

Parameters
  rectangle Rectangle to test

Returns
  True if rectangles overlap, false otherwise

See also
  contains

---

template<typename T>

bool sf::Rect<T>::intersects ( const Rect<T> & rectangle, Rect<T> & intersection ) const

Check the intersection between two rectangles.

This overload returns the overlapped rectangle in the intersection parameter.
Parameters
- rectangle  Rectangle to test
- intersection  Rectangle to be filled with the intersection

Returns
- True if rectangles overlap, false otherwise

See also
- contains
template<typename T >
bool operator!= ( const Rect<T> & left, 
                 const Rect<T> & right )

Overload of binary operator !=.
This operator compares strict difference between two rectangles.

Parameters
  left  Left operand (a rectangle)
  right Right operand (a rectangle)

Returns
  True if left is not equal to right

template<typename T >
bool operator== ( const Rect<T> & left, 
                 const Rect<T> & right )

Overload of binary operator ==.
This operator compares strict equality between two rectangles.

Parameters
**left**  Left operand (a rectangle)
**right** Right operand (a rectangle)

**Returns**
True if **left** is equal to **right**
<table>
<thead>
<tr>
<th>Template Type</th>
<th>Description</th>
<th>Definition Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sf::Rect&lt;T&gt;::height</code></td>
<td>Height of the rectangle.</td>
<td>Line 160, file <code>Rect.hpp</code></td>
</tr>
<tr>
<td><code>sf::Rect&lt;T&gt;::left</code></td>
<td>Left coordinate of the rectangle.</td>
<td>Line 157, file <code>Rect.hpp</code></td>
</tr>
<tr>
<td><code>sf::Rect&lt;T&gt;::top</code></td>
<td>Top coordinate of the rectangle.</td>
<td>Line 158, file <code>Rect.hpp</code></td>
</tr>
<tr>
<td><code>sf::Rect&lt;T&gt;::width</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Width of the rectangle.

Definition at line 159 of file Rect.hpp.

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

- Rect.hpp
sf::RectangleShape Class Reference

Graphics module

Specialized shape representing a rectangle. More...

#include <RectangleShape.hpp>

Inheritance diagram for sf::RectangleShape:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RectangleShape</strong> (const Vector2f &amp;size=Vector2f(0, 0))</td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void setSize (const Vector2f &amp;size)</strong></td>
<td>Set the size of the rectangle. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const Vector2f &amp; getSize () const</strong></td>
<td>Get the size of the rectangle. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>virtual std::size_t getPointCount () const</strong></td>
<td>Get the number of points defining the shape. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>virtual Vector2f getPoint (std::size_t index) const</strong></td>
<td>Get a point of the rectangle. <a href="#">More...</a></td>
</tr>
<tr>
<td>*<em>void setTexture (const Texture <em>texture, bool resetRect=false)</em></em></td>
<td>Change the source texture of the shape. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void setTextureRect (const IntRect &amp;rect)</strong></td>
<td>Set the sub-rectangle of the texture that the shape will display.</td>
</tr>
<tr>
<td><strong>void setFillColor (const Color &amp;color)</strong></td>
<td>Set the fill color of the shape. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void setOutlineColor (const Color &amp;color)</strong></td>
<td>Set the outline color of the shape. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void setOutlineThickness (float thickness)</strong></td>
<td>Set the thickness of the shape's outline. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const Texture * getTexture () const</strong></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>const IntRect &amp; getTextureRect () const</code></td>
<td>Get the source texture of the shape. More...</td>
</tr>
<tr>
<td><code>const Color &amp; getFillColor () const</code></td>
<td>Get the fill color of the shape. More...</td>
</tr>
<tr>
<td><code>const Color &amp; getOutlineColor () const</code></td>
<td>Get the outline color of the shape. More...</td>
</tr>
<tr>
<td><code>float getOutlineThickness () const</code></td>
<td>Get the outline thickness of the shape. More...</td>
</tr>
<tr>
<td><code>FloatRect getLocalBounds () const</code></td>
<td>Get the local bounding rectangle of the entity. More..</td>
</tr>
<tr>
<td><code>FloatRect getGlobalBounds () const</code></td>
<td>Get the global (non-minimal) bounding rectangle of the entity. More..</td>
</tr>
<tr>
<td><code>void setPosition (float x, float y)</code></td>
<td>set the position of the object More...</td>
</tr>
<tr>
<td><code>void setPosition (const Vector2f &amp;position)</code></td>
<td>set the position of the object More...</td>
</tr>
<tr>
<td><code>void setRotation (float angle)</code></td>
<td>set the orientation of the object More...</td>
</tr>
<tr>
<td><code>void setScale (float factorX, float factorY)</code></td>
<td>set the scale factors of the object More...</td>
</tr>
<tr>
<td><code>void setScale (const Vector2f &amp;factors)</code></td>
<td>set the scale factors of the object More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>void setOrigin(float x, float y)</code></td>
<td>set the local origin of the object</td>
</tr>
<tr>
<td><code>void setOrigin(const Vector2f &amp;origin)</code></td>
<td>set the local origin of the object</td>
</tr>
<tr>
<td><code>const Vector2f &amp; getPosition()</code></td>
<td>get the position of the object</td>
</tr>
<tr>
<td><code>float getRotation()</code></td>
<td>get the orientation of the object</td>
</tr>
<tr>
<td><code>const Vector2f &amp; getScale()</code></td>
<td>get the current scale of the object</td>
</tr>
<tr>
<td><code>const Vector2f &amp; getOrigin()</code></td>
<td>get the local origin of the object</td>
</tr>
<tr>
<td><code>void move(float offsetX, float offsetY)</code></td>
<td>Move the object by a given offset.</td>
</tr>
<tr>
<td><code>void move(const Vector2f &amp;offset)</code></td>
<td>Move the object by a given offset.</td>
</tr>
<tr>
<td><code>void rotate(float angle)</code></td>
<td>Rotate the object.</td>
</tr>
<tr>
<td><code>void scale(float factorX, float factorY)</code></td>
<td>Scale the object.</td>
</tr>
<tr>
<td><code>void scale(const Vector2f &amp;factor)</code></td>
<td>Scale the object.</td>
</tr>
<tr>
<td><code>const Transform &amp; getTransform()</code></td>
<td>get the combined transform of the object</td>
</tr>
</tbody>
</table>
const Transform & getInverseTransform () const
get the inverse of the combined transform of the obj
Protected Member Functions

| void update () | Recompute the internal geometry of the shape. More... |
Detailed Description

Specialized shape representing a rectangle.

This class inherits all the functions of `sf::Transformable` (position, rotation functions of `sf::Shape` (outline, color, texture, ...).

Usage example:

```cpp
sf::RectangleShape rectangle;
rectangle.setSize(sf::Vector2f(100, 50));
rectangle.setOutlineColor(sf::Color::Red);
rectangle.setOutlineThickness(5);
rectangle.setPosition(10, 20);
...
window.draw(rectangle);
```

See also

`sf::Shape`, `sf::CircleShape`, `sf::ConvexShape`

Definition at line 41 of file `RectangleShape.hpp`. 
sf::RectangleShape::RectangleShape (const Vector2f & size = Vector2f(1, 1))

Default constructor.

**Parameters**
- **size** Size of the rectangle
const Color& sf::Shape::getFillColor() const

Get the fill color of the shape.

**Returns**
Fill color of the shape

**See also**
setFillColor

---

FloatRect sf::Shape::getGlobalBounds() const

Get the global (non-minimal) bounding rectangle of the entity.

The returned rectangle is in global coordinates, which means that
transformations (translation, rotation, scale, ...) that are applied to the entity returns the bounds of the shape in the global 2D world's coordinate system.

This function does not necessarily return the *minimal* bounding rectangle. The returned rectangle covers all the vertices (but possibly more). This allows it to be a fast approximation as a first check; you may want to use more precise checks on top.

**Returns**
Global bounding rectangle of the entity
const Transform& sf::Transformable::getInverseTransform() const

get the inverse of the combined transform of the object

**Returns**
Inverse of the combined transformations applied to the object

**See also**
getTransform

**FloatRect** sf::Shape::getLocalBounds() const

Get the local bounding rectangle of the entity.

The returned rectangle is in local coordinates, which means that (translation, rotation, scale, ...) that are applied to the entity. In other bounds of the entity in the entity's coordinate system.

**Returns**
Local bounding rectangle of the entity

**const Vector2f**& sf::Transformable::getOrigin() const

get the local origin of the object

**Returns**
Current origin

**See also**
setOrigin
const Color& sf::Shape::getOutlineColor() const

Get the outline color of the shape.

**Returns**
Outline color of the shape

**See also**
setOutlineColor

float sf::Shape::getOutlineThickness() const

Get the outline thickness of the shape.

**Returns**
Outline thickness of the shape

**See also**
setOutlineThickness

virtual Vector2f sf::RectangleShape::getPoint(std::size_t index) const

Get a point of the rectangle.

The returned point is in local coordinates, that is, the shape's transforms taken into account. The result is undefined if `index` is out of the valid range.

**Parameters**

- `index` Index of the point to get, in range [0 .. 3]
**Returns**

index-th point of the shape

Implements sf::Shape.

---

**virtual std::size_t sf::RectangleShape::getPointCount ( ) const**

Get the number of points defining the shape.

**Returns**

Number of points of the shape. For rectangle shapes, this number is

Implements sf::Shape.

---

**const Vector2f& sf::Transformable::getPosition ( ) const**

get the position of the object

**Returns**

Current position

**See also**

setPosition

---

**float sf::Transformable::getRotation ( ) const**

get the orientation of the object

The rotation is always in the range [0, 360].
**Returns**
Current rotation, in degrees

**See also**
setRotation

---

**const Vector2f sf::Transformable::getScale() const**

get the current scale of the object

**Returns**
Current scale factors

**See also**
setScale

---

**const Vector2f sf::RectangleShape::getSize() const**

Get the size of the rectangle.

**Returns**
Size of the rectangle

**See also**
setSize

---

**const Texture* sf::Shape::getTexture() const**

Get the source texture of the shape.
If the shape has no source texture, a NULL pointer is returned. The returned pointer is const, which means that you can't modify the texture when you retrieve it with this function.

**Returns**
Pointer to the shape's texture

**See also**
setTexture

---

**const IntRect**& sf::Shape::getTextureRect ( ) const

Get the sub-rectangle of the texture displayed by the shape.

**Returns**
Texture rectangle of the shape

**See also**
setTextureRect

---

**const Transform**& sf::Transformable::getTransform ( ) const

get the combined transform of the object

**Returns**
Transform combining the position/rotation/scale/origin of the object

**See also**
getInverseTransform

---

void sf::Transformable::move ( float offsetX,
Move the object by a given offset.

This function adds to the current position of the object, unlike `setPosition` which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f pos = object.getPosition();
object.setPosition(pos.x + offsetX, pos.y + offsetY);
```

**Parameters**
- `offsetX` X offset
- `offsetY` Y offset

**See also**
- `setPosition`

---

```cpp
void sf::Transformable::move (const Vector2f & offset)
```

Move the object by a given offset.

This function adds to the current position of the object, unlike `setPosition` which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setPosition(object.getPosition() + offset);
```

**Parameters**
- `offset` Offset

**See also**
- `setPosition`
void sf::Transformable::rotate ( float angle )

Rotate the object.

This function adds to the current rotation of the object, unlike setRotation which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setRotation(object.getRotation() + angle);
```

**Parameters**

- **angle** Angle of rotation, in degrees

---

void sf::Transformable::scale ( float factorX, float factorY )

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factorX, scale.y * factorY);
```

**Parameters**

- **factorX** Horizontal scale factor
- **factorY** Vertical scale factor

**See also**

setScale
### void sf::Transformable::scale ( const Vector2f & factor )

Scale the object.

This function multiplies the current scale of the object, unlike `setScale` which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factor.x, scale.y * factor.y);
```

**Parameters**
- **factor** Scale factors

**See also**
- `setScale`

### void sf::Shape::setFillColor ( const Color & color )

Set the fill color of the shape.

This color is modulated (multiplied) with the shape's texture if any. It can colorize the shape or change its global opacity. You can use `sf::Color::Transparent` to make the inside of the shape transparent, and have the outline alone. By default, the shape's fill color is opaque white.

**Parameters**
- **color** New color of the shape

**See also**
- `getFillColor`, `setOutlineColor`
### `void sf::Transformable::setOrigin(float x, float y)`

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object, and ignore all transformations. The default origin of a transformable object is `(0, 0)`.

**Parameters**
- **x** X coordinate of the new origin
- **y** Y coordinate of the new origin

**See also**
- `getOrigin`

### `void sf::Transformable::setOrigin(const Vector2f & origin)`

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object, and ignore all transformations. The default origin of a transformable object is `(0, 0)`.

**Parameters**
- **origin** New origin

**See also**
- `getOrigin`
void sf::Shape::setOutlineColor ( const Color & color )

Set the outline color of the shape.

By default, the shape's outline color is opaque white.

Parameters
  color New outline color of the shape

See also
  getOutlineColor, setFillColor

void sf::Shape::setOutlineThickness ( float thickness )

Set the thickness of the shape's outline.

Note that negative values are allowed (so that the outline expands towards using zero disables the outline. By default, the outline thickness is 0.

Parameters
  thickness New outline thickness

See also
  getOutlineThickness

void sf::Transformable::setPosition ( float x,
                                      float y )

set the position of the object
This function completely overwrites the previous position. See the move function on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**

- **x** X coordinate of the new position
- **y** Y coordinate of the new position

**See also**

- move, getPosition

```cpp
void sf::Transformable::setPosition(const Vector2f & position)
```

set the position of the object

This function completely overwrites the previous position. See the move function on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**

- **position** New position

**See also**

- move, getPosition

```cpp
void sf::Transformable::setRotation(float angle)
```

set the orientation of the object

This function completely overwrites the previous rotation. See the rotate function on the previous rotation instead. The default rotation of a transformable object is 0.

**Parameters**
angle New rotation, in degrees

See also
rotate, getRotation

void sf::Transformable::setScale ( float factorX,
                                   float factorY
                               )

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function instead. The default scale of a transformable object is

Parameters
factorX New horizontal scale factor
factorY New vertical scale factor

See also
scale, getScale

void sf::Transformable::setScale ( const Vector2f & factors )

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function instead. The default scale of a transformable object is

Parameters
factors New scale factors
**See also**

scale, getSize

---

### void sf::RectangleShape::setSize (const Vector2f & size)

Set the size of the rectangle.

**Parameters**

- **size** New size of the rectangle

**See also**

getSize

---

### void sf::Shape::setTexture (const Texture * texture, bool resetRect = false)

Change the source texture of the shape.

The `texture` argument refers to a texture that must exist as long as the shape doesn’t store its own copy of the texture, but rather keeps a pointer to it. If the source texture is destroyed and the shape tries to use it, `texture` can be NULL to disable texturing. If `resetRect` is true, the TextureRect property of the shape is automatically adjusted to the size of the new texture. If it is false, the texture rect is left unchanged.

**Parameters**

- **texture** New texture
- **resetRect** Should the texture rect be reset to the size of the new texture?

**See also**

getTexture, setTextureRect
void sf::Shape::setTextureRect ( const IntRect & rect )

Set the sub-rectangle of the texture that the shape will display.

The texture rect is useful when you don't want to display the whole texture. By default, the texture rect covers the entire texture.

Parameters

- rect Rectangle defining the region of the texture to display

See also

- getTextureRect, setTexture

void sf::Shape::update ( )

Recompute the internal geometry of the shape.

This function must be called by the derived class every time the shape's points change (i.e. the result of either getPointCount or getPoint is different).

The documentation for this class was generated from the following file:
- RectangleShape.hpp
sf::RenderStates Class Reference

Define the states used for drawing to a RenderTarget. More...

#include <RenderStates.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RenderStates()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>RenderStates(const BlendMode &amp;theBlendMode)</code></td>
<td>Construct a default set of render states with a custom blend mode. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>RenderStates(const Transform &amp;theTransform)</code></td>
<td>Construct a default set of render states with a custom transform. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>RenderStates(const Texture *theTexture)</code></td>
<td>Construct a default set of render states with a custom texture. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>RenderStates(const Shader *theShader)</code></td>
<td>Construct a default set of render states with a custom shader. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>RenderStates(const BlendMode &amp;theBlendMode, const Transform &amp;theTransform, const Texture *theTexture, const Shader *theShader)</code></td>
<td>Construct a set of render states with all its attributes. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Public Attributes

<table>
<thead>
<tr>
<th>Public Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BlendMode</strong></td>
<td>blendMode, Blending mode. More...</td>
</tr>
<tr>
<td><strong>Transform</strong></td>
<td>transform, Transform. More...</td>
</tr>
<tr>
<td><strong>const Texture</strong> *</td>
<td>texture, Texture. More...</td>
</tr>
<tr>
<td><strong>const Shader</strong> *</td>
<td>shader, Shader. More...</td>
</tr>
</tbody>
</table>
### Static Public Attributes

<table>
<thead>
<tr>
<th>static const RenderStates Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special instance holding the default render states.</td>
</tr>
</tbody>
</table>
Detailed Description

Define the states used for drawing to a `RenderTarget`.

There are four global states that can be applied to the drawn objects:

- the blend mode: how pixels of the object are blended with the background
- the transform: how the object is positioned/rotated/scaled
- the texture: what image is mapped to the object
- the shader: what custom effect is applied to the object

High-level objects such as sprites or text force some of these states when they are drawn. For example, a sprite will set its own texture, so that you don't have to care about it when drawing it.

The transform is a special case: sprites, texts and shapes (and it's a good idea to do it with your own drawable classes too) combine their transform with the one that is passed in. So that you can use a "global" transform on top of each object's transform.

Most objects, especially high-level drawables, can be drawn directly without defining render states explicitly – the default set of states is ok in most cases.

```cpp
window.draw(sprite);
```

If you want to use a single specific render state, for example a shader, you can pass it directly to the `Draw` function: `sf::RenderStates` has an implicit one-argument constructor for each state.

```cpp
window.draw(sprite, shader);
```

When you're inside the `Draw` function of a drawable object (inherited from `Drawable`), you can pass the render states unmodified, or change some of them. For example:
combine the current transform with its own transform. A sprite will set its texture.

**See also**
- sf::RenderTarget, sf::Drawable

Definition at line 45 of file `RenderStates.hpp`. 
Constructor & Destructor Documentation

sf::RenderStates::RenderStates ( )

Default constructor.

Constructing a default set of render states is equivalent to using sf::RenderStates:

- the BlendAlpha blend mode
- the identity transform
- a null texture
- a null shader

sf::RenderStates::RenderStates ( const BlendMode & theBlendMode )

Construct a default set of render states with a custom blend mode.

Parameters

theBlendMode Blend mode to use

sf::RenderStates::RenderStates ( const Transform & theTransform )

Construct a default set of render states with a custom transform.

Parameters
theTransform Transform to use

sf::RenderStates::RenderStates ( const Texture * theTexture )

Construct a default set of render states with a custom texture.

Parameters
  theTexture Texture to use

sf::RenderStates::RenderStates ( const Shader * theShader )

Construct a default set of render states with a custom shader.

Parameters
  theShader Shader to use

sf::RenderStates::RenderStates ( const BlendMode & theBlendMode,
    const Transform & theTransform,
    const Texture * theTexture,
    const Shader * theShader )

Construct a set of render states with all its attributes.

Parameters
  theBlendMode Blend mode to use
  theTransform Transform to use
  theTexture Texture to use
  theShader Shader to use
**Member Data Documentation**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BlendMode sf::RenderStates::blendMode</strong></td>
<td>Blending mode.</td>
</tr>
<tr>
<td></td>
<td>Definition at line 115 of file <code>RenderStates.hpp</code>.</td>
</tr>
<tr>
<td><strong>const RenderStates sf::RenderStates::Default</strong></td>
<td>Special instance holding the default render states.</td>
</tr>
<tr>
<td></td>
<td>Definition at line 110 of file <code>RenderStates.hpp</code>.</td>
</tr>
<tr>
<td><em><em>const Shader</em> sf::RenderStates::shader</em>*</td>
<td>Shader.</td>
</tr>
<tr>
<td></td>
<td>Definition at line 118 of file <code>RenderStates.hpp</code>.</td>
</tr>
<tr>
<td><em><em>const Texture</em> sf::RenderStates::texture</em>*</td>
<td>Texture.</td>
</tr>
<tr>
<td></td>
<td>Definition at line 117 of file <code>RenderStates.hpp</code>.</td>
</tr>
</tbody>
</table>
Transform

Transform.

Definition at line 116 of file RenderStates.hpp.

The documentation for this class was generated from the following file:
- RenderStates.hpp
sf::RenderTarget Class Reference

Base class for all render targets (window, texture, ...)

```
#include <RenderTarget.hpp>
```

Inheritance diagram for sf::RenderTarget:
## Public Member Functions

<table>
<thead>
<tr>
<th>virtual</th>
<th>~RenderTarget ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destructor. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>clear (const Color &amp;color=Color(0, 0, 0, 255))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear the entire target with a single color. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setView (const View &amp;view)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the current active view. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>const View &amp;</th>
<th>getView () const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the view currently in use in the render target. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>const View &amp;</th>
<th>getDefaultView () const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the default view of the render target. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IntRect</th>
<th>getViewport (const View &amp;view) const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the viewport of a view, applied to this render target.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vector2f</th>
<th>mapPixelToCoords (const Vector2i &amp;point) const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert a point from target coordinates to world coordinates. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vector2f</th>
<th>mapPixelToCoords (const Vector2i &amp;point, const View &amp;view)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert a point from target coordinates to world coordinates.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vector2i</th>
<th>mapCoordsToPixel (const Vector2f &amp;point) const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert a point from world coordinates to target coordinates. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vector2i</th>
<th>mapCoordsToPixel (const Vector2f &amp;point, const View &amp;view)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert a point from world coordinates to target coordinates.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>draw (const Drawable &amp;drawable, const RenderStates &amp;states=Draw)</code></td>
<td>Draw a drawable object to the render target.</td>
</tr>
<tr>
<td><code>draw (const Vertex *vertices, std::size_t vertexCount, RenderStates &amp;states=RenderStates::Default)</code></td>
<td>Draw primitives defined by an array of vertices.</td>
</tr>
<tr>
<td><code>virtual Vector2u getSize () const =0</code></td>
<td>Return the size of the rendering region of the target.</td>
</tr>
<tr>
<td><code>pushGLStates ()</code></td>
<td>Save the current OpenGL render states and matrices.</td>
</tr>
<tr>
<td><code>popGLStates ()</code></td>
<td>Restore the previously saved OpenGL render states and matrices.</td>
</tr>
<tr>
<td><code>resetGLStates ()</code></td>
<td>Reset the internal OpenGL states so that the target is ready for drawing.</td>
</tr>
</tbody>
</table>
# Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RenderTarget()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>initialize()</code></td>
<td>Performs the common initialization step after creation. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Base class for all render targets (window, texture, ...)

sf::RenderTarget defines the common behavior of all the 2D render targets.

It makes it possible to draw 2D entities like sprites, shapes, text without directly.

A sf::RenderTarget is also able to use views (sf::View), which are a kind of 2D cameras. With views you can globally scroll, rotate or zoom everything that is drawn, without having to transform every single entity. See the documentation of sf::View for more details and sample pieces of code.

On top of that, render targets are still able to render direct OpenGL stuff. OpenGL calls and regular SFML drawing commands. When doing so, make sure that OpenGL states are not messed up by calling the pushGLStates/popGLStates functions.

See also

sf::RenderWindow, sf::RenderTexture, sf::View

Definition at line 51 of fileRenderTarget.hpp.
### Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual sf::RenderTarget::~RenderTarget ( )</code></td>
<td>Destructor.</td>
</tr>
<tr>
<td><code>sf::RenderTarget::RenderTarget ( )</code></td>
<td>Default constructor.</td>
</tr>
</tbody>
</table>
Member Function Documentation

**void sf::RenderTarget::clear ( const Color & color = Color(0, 0, 0, 255) )**

Clear the entire target with a single color.

This function is usually called once every frame, to clear the previous contents.

**Parameters**
- **color** Fill color to use to clear the render target

**void sf::RenderTarget::draw ( const Drawable & drawable, const RenderStates & states = RenderStates::Default )**

Draw a drawable object to the render target.

**Parameters**
- **drawable** Object to draw
- **states** Render states to use for drawing

**void sf::RenderTarget::draw ( const Vertex * vertices, std::size_t vertexCount, const RenderStates & states = RenderStates::Default )**

...
Draw primitives defined by an array of vertices.

**Parameters**

- **vertices**: Pointer to the vertices
- **vertexCount**: Number of vertices in the array
- **type**: Type of primitives to draw
- **states**: Render states to use for drawing

### `const View& sf::RenderTarget::getDefaultView ( ) const`

Get the default view of the render target.

The default view has the initial size of the render target, and never changes after the target has been created.

**Returns**

- The default view of the render target

**See also**

- `setView`, `getView`

### `virtual Vector2u sf::RenderTarget::getSize ( ) const`

Return the size of the rendering region of the target.

**Returns**

- Size in pixels

Implemented in `sf::RenderTexture`, and `sf::RenderWindow`. 
**const View** & sf::RenderTarget::getView ( ) const

Get the view currently in use in the render target.

**Returns**

The view object that is currently used

**See also**

setView, getDefaultView

---

**IntRect sf::RenderTarget::getViewport ( const View & view ) const**

Get the viewport of a view, applied to this render target.

The viewport is defined in the view as a ratio, this function simply adjusts the dimensions of the render target to calculate the pixels rectangle that the viewport actually covers in the target.

**Parameters**

- **view** The view for which we want to compute the viewport

**Returns**

Viewport rectangle, expressed in pixels

---

**void sf::RenderTarget::initialize ( )**

Performs the common initialization step after creation.

The derived classes must call this function after the target is created and
Vector2i sf::RenderTarget::mapCoordsToPixel (const Vector2f & point)

Convert a point from world coordinates to target coordinates, using the current view.

This function is an overload of the mapCoordsToPixel function that implicitly uses the current view. It is equivalent to:

```
target.mapCoordsToPixel(point, target.getView());
```

**Parameters**

- **point** Point to convert

**Returns**

The converted point, in target coordinates (pixels)

**See also**

mapPixelToCoords

Vector2i sf::RenderTarget::mapCoordsToPixel (const Vector2f & point, const View & view)

Convert a point from world coordinates to target coordinates.

This function finds the pixel of the render target that matches the given point, through the same process as the graphics card, to compute the final position.

Initially, both coordinate systems (world units and target pixels) match perfectly. But if you define a custom view or resize your render target, this assertion is not true anymore. For example, a point located at (150, 75) in your 2D world may map to the pixel (10, 50) of your render target if the view is translated by (140, 25).
This version uses a custom view for calculations, see the other overload to use the current view of the render target.

**Parameters**

- **point** Point to convert
- **view** The view to use for converting the point

**Returns**

The converted point, in target coordinates (pixels)

**See also**

- `mapPixelToCoords`

---

**Vector2f sf::RenderTarget::mapPixelToCoords (const Vector2i & point)**

Convert a point from target coordinates to world coordinates, using the current view.

This function is an overload of the `mapPixelToCoords` function that implicitly uses the current view.

Equivalent to:

```
target.mapPixelToCoords(point, target.getView());
```

**Parameters**

- **point** Pixel to convert

**Returns**

The converted point, in "world" coordinates

**See also**

- `mapCoordsToPixel`
Vector2f sf::RenderTarget::mapPixelToCoords ( const Vector2i & point, const View & view ) const

Convert a point from target coordinates to world coordinates.

This function finds the 2D position that matches the given pixel of the render target, which does the inverse of what the graphics card does, to find the initial position of a rendered pixel.

Initially, both coordinate systems (world units and target pixels) match perfectly. If you define a custom view or resize your render target, this assertion is not true anymore. For example, a point located at (10, 50) in your render target may map to the point (150, 75) in your 2D world if the view is translated by (140, 25).

For render-windows, this function is typically used to find which point is located below the mouse cursor.

This version uses a custom view for calculations, see the other overload if you want to use the current view of the render target.

**Parameters**
- **point** Pixel to convert
- **view** The view to use for converting the point

**Returns**
The converted point, in "world" units

**See also**
- mapCoordsToPixel

void sf::RenderTarget::popGLStates ( )
Restore the previously saved OpenGL render states and matrices.

See the description of pushGLStates to get a detailed description of these.

**See also**

*pushGLStates*

---

**void sf::RenderTarget::pushGLStates ()**

Save the current OpenGL render states and matrices.

This function can be used when you mix SFML drawing and direct OpenGL rendering, it ensures that:

- SFML's internal states are not messed up by your OpenGL code
- your OpenGL states are not modified by a call to a SFML function

More specifically, it must be used around code that calls Draw functions.

```cpp
// OpenGL code here...
window.pushGLStates();
window.draw(...);
window.draw(...);
window.popGLStates();
// OpenGL code here...
```

Note that this function is quite expensive: it saves all the possible OpenGL states you don't care about. Therefore it should be used wisely. It is probably best results will be achieved if you handle OpenGL states yourself (because they really changed, and need to be saved and restored). Take a look at the [resetGLStates](#) function if you do so.

**See also**
Reset the internal OpenGL states so that the target is ready for drawing.

This function can be used when you mix SFML drawing and direct OpenGL rendering, if you choose not to use pushGLStates/popGLStates. It makes sure that all OpenGL states that subsequent `draw()` calls will work as expected.

Example:

```cpp
// OpenGL code here...
glPushAttrib(...);
window.resetGLStates();
window.draw(...);
window.draw(...);
gxPopAttrib(...);
// OpenGL code here...
```

Change the current active view.

The view is like a 2D camera, it controls which part of the 2D scene is visible in the render target. The new view will affect everything that is drawn, until another view is set. The render target keeps its own copy of the view object, so it is not necessary to call this function. To restore the original view of the target, you can pass the result of calling this function.

**Parameters**

- `view` New view to use

See also
getView, getDefaultView

The documentation for this class was generated from the following file:
- RenderTarget.hpp
Target for off-screen 2D rendering into a texture. More...

#include <RenderTexture.hpp>

Inheritance diagram for sf::RenderTexture:
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RenderTexture ()</td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual ~RenderTexture ()</td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool create (unsigned int width, unsigned int height, bool depthBuffer=false)</td>
<td>Create the render-texture. <a href="#">More...</a></td>
</tr>
<tr>
<td>void setSmooth (bool smooth)</td>
<td>Enable or disable texture smoothing. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool isSmooth () const</td>
<td>Tell whether the smooth filtering is enabled or not. <a href="#">More...</a></td>
</tr>
<tr>
<td>void setRepeated (bool repeated)</td>
<td>Enable or disable texture repeating. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool isRepeated () const</td>
<td>Tell whether the texture is repeated or not. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool generateMipmap ()</td>
<td>Generate a mipmap using the current texture data. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool setActive (bool active=true)</td>
<td>Activate or deactivate the render-texture for rendering. <a href="#">More...</a></td>
</tr>
<tr>
<td>void display ()</td>
<td>Update the contents of the target texture. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual Vector2u getSize () const</td>
<td></td>
</tr>
</tbody>
</table>
Return the size of the rendering region of the texture. More...

```cpp
const Texture & getTexture () const
Get a read-only reference to the target texture. More...
```

```cpp
void clear (const Color &color=Color(0, 0, 0, 255))
Clear the entire target with a single color. More...
```

```cpp
void setView (const View &view)
Change the current active view. More...
```

```cpp
const View & getView () const
Get the view currently in use in the render target. More.
```

```cpp
const View & getDefaultView () const
Get the default view of the render target. More...
```

```cpp
IntRect getViewport (const View &view) const
Get the viewport of a view, applied to this render target.
```

```cpp
Vector2f mapPixelToCoords (const Vector2i &point) const
Convert a point from target coordinates to world coordinates. More...
```

```cpp
Vector2f mapPixelToCoords (const Vector2i &point, const View &view)
Convert a point from target coordinates to world coordinates.
```

```cpp
Vector2i mapCoordsToPixel (const Vector2f &point) const
Convert a point from world coordinates to target coordinates. More...
```

```cpp
Vector2i mapCoordsToPixel (const Vector2f &point, const View &view)
Convert a point from world coordinates to target coordinates.
```

```cpp
void draw (const Drawable &drawable, const RenderStates &states=
```
Draw a drawable object to the render target. **More...**

<table>
<thead>
<tr>
<th>void draw (const Vertex *vertices, std::size_t vertexCount, PrimitiveType RenderStates &amp;states=RenderStates::Default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw primitives defined by an array of vertices. <strong>More...</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void pushGLStates ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save the current OpenGL render states and matrices. <strong>More...</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void popGLStates ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restore the previously saved OpenGL render states and matrices. <strong>More...</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void resetGLStates ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset the internal OpenGL states so that the target is ready for drawing. <strong>More...</strong></td>
</tr>
</tbody>
</table>
Protected Member Functions

```c
void initialize()
Performs the common initialization step after creation. More...
```
Detailed Description

Target for off-screen 2D rendering into a texture.

sf::RenderTexture is the little brother of sf::RenderWindow.

It implements the same 2D drawing and OpenGL-related functions (see for more details), the difference is that the result is stored in an off-screen a window.

Rendering to a texture can be useful in a variety of situations:

- precomputing a complex static texture (like a level's background from tiles)
- applying post-effects to the whole scene with shaders
- creating a sprite from a 3D object rendered with OpenGL
- etc.

Usage example:

```cpp
// Create a new render-window
sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");

// Create a new render-texture
sf::RenderTexture texture;
if (!texture.create(500, 500))
    return -1;

// The main loop
while (window.isOpen())
{
    // Event processing
    // ...

    // Clear the whole texture with red color
    texture.clear(sf::Color::Red);

    // Draw stuff to the texture
    texture.draw(sprite);  // sprite is a sf::Sprite
```
texture.draw(shape); // shape is a sf::Shape
texture.draw(text); // text is a sf::Text

// We're done drawing to the texture
texture.display();

// Now we start rendering to the window, clear it first
window.clear();

// Draw the texture
sf::Sprite sprite(texture.getTexture());
window.draw(sprite);

// End the current frame and display its contents on screen
window.display();

Like sf::RenderWindow, sf::RenderTexture is still able to render direct OpenGL calls and regular SFML drawing commands. If you need a depth buffer for 3D rendering, don't forget to request it when calling RenderTexture::create.

See also

sf::RenderTarget, sf::RenderWindow, sf::View, sf::Texture

Definition at line 47 of file RenderTexture.hpp.
Constructor & Destructor Documentation

sf::RenderTexture::RenderTexture ( )

Default constructor.

Constructs an empty, invalid render-texture. You must call create to have

See also
 create

virtual sf::RenderTexture::~RenderTexture ( )

Destructor.
Member Function Documentation

```cpp
text
void sf::RenderTarget::clear ( const Color & color = Color(0, 0, 0, 255) )
```

Clear the entire target with a single color.

This function is usually called once every frame, to clear the previous contents.

**Parameters**

- `color` Fill color to use to clear the render target

```cpp
text
bool sf::RenderTexture::create ( unsigned int width, unsigned int height, bool depthBuffer = false )
```

Create the render-texture.

Before calling this function, the render-texture is in an invalid state, thus doing anything with the render-texture. The last parameter, `depthBuffer`, render-texture for 3D OpenGL rendering that requires a depth buffer. Otherwise you should leave this parameter to false (which is its default value).

**Parameters**

- `width` Width of the render-texture
- `height` Height of the render-texture
- `depthBuffer` Do you want this render-texture to have a depth buffer
<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>True if creation has been successful</td>
</tr>
</tbody>
</table>

### void sf::RenderTexture::display ( )

Update the contents of the target texture.

This function updates the target texture with what has been drawn so far. This function is mandatory at the end of rendering. Not calling it may leave the texture in an undefined state.

### void sf::RenderTarget::draw ( const Drawable & drawable, const RenderStates & states = RenderStates::Default )

Draw a drawable object to the render target.

**Parameters**

- **drawable** Object to draw
- **states** Render states to use for drawing

### void sf::RenderTarget::draw ( const Vertex * vertices, std::size_t vertexCount, PrimitiveType type, const RenderStates & states = RenderStates::Default )

Draw primitives defined by an array of vertices.

**Parameters**
vertices | Pointer to the vertices
vertexCount | Number of vertices in the array
type | Type of primitives to draw
states | Render states to use for drawing

bool sf::RenderTexture::generateMipmap ()

Generate a mipmap using the current texture data.

This function is similar to Texture::generateMipmap and operates on the current texture used for drawing. Be aware that any draw operation may modify the base level of the texture. Calling this function only makes sense after all drawing is completed and display has been called. Calling display after subsequent drawing will lead to undefined behavior if a mipmap had been previously generated.

Returns
True if mipmap generation was successful, false if unsuccessful.

const View& sf::RenderTarget::getDefaultView () const

Get the default view of the render target.

The default view has the initial size of the render target, and never changes after the target has been created.

Returns
The default view of the render target.

See also
setView, getView
virtual Vector2u sf::RenderTexture::getSize () const

Return the size of the rendering region of the texture.
The returned value is the size that you passed to the create function.

**Returns**
Size in pixels

Implements sf::RenderTarget.

const Texture& sf::RenderTexture::getTexture () const

Get a read-only reference to the target texture.

After drawing to the render-texture and calling Display, you can retrieve the function, and draw it using a sprite (for example). The internal sf::Texture the same instance, so that it is possible to call this function once and even after it is modified.

**Returns**
Const reference to the texture

const View& sf::RenderTarget::getView () const

Get the view currently in use in the render target.

**Returns**
The view object that is currently used
See also

`setView`, `getDefaultView`

## `IntRect sf::RenderTarget::getViewport (const View & view) const`

Get the viewport of a view, applied to this render target.

The viewport is defined in the view as a ratio, this function simply applies the ratio to the dimensions of the render target to calculate the pixels rectangle that the viewport actually covers in the target.

### Parameters

- `view` The view for which we want to compute the viewport

### Returns

Viewport rectangle, expressed in pixels

## `void sf::RenderTarget::initialize ( )`

Performs the common initialization step after creation.

The derived classes must call this function after the target is created and ready for drawing.

## `bool sf::RenderTexture::isRepeated ( ) const`

Tell whether the texture is repeated or not.

### Returns

- True if texture is repeated
bool sf::RenderTexture::isSmooth ( ) const

Tell whether the smooth filtering is enabled or not.

**Returns**

True if texture smoothing is enabled

**See also**

setSmooth

---

Vector2i sf::RenderTarget::mapCoordsToPixel ( const Vector2f & point )

Convert a point from world coordinates to target coordinates, using the current view.

This function is an overload of the mapCoordsToPixel function that implicitly uses the current view. It is equivalent to:

```cpp
target.mapCoordsToPixel(point, target.getView());
```

**Parameters**

- **point** Point to convert

**Returns**

The converted point, in target coordinates (pixels)

**See also**

mapPixelToCoords
Convert a point from world coordinates to target coordinates.

This function finds the pixel of the render target that matches the given through the same process as the graphics card, to compute the final pos

Initially, both coordinate systems (world units and target pixels) matc
custom view or resize your render target, this assertion is not true anym
75) in your 2D world may map to the pixel (10, 50) of your render tar
target (140, 25).

This version uses a custom view for calculations, see the other overload if you use the current view of the render target.

**Parameters**

- `point` Point to convert
- `view` The view to use for converting the point

**Returns**

The converted point, in target coordinates (pixels)

**See also**

mapPixelToCoords

---

Convert a point from target coordinates to world coordinates, using the c
This function is an overload of the mapPixelToCoords function that is equivalent to:

```cpp
target.mapPixelToCoords(point, target.getView());
```

**Parameters**

- **point** Pixel to convert

**Returns**

The converted point, in "world" coordinates

**See also**

mapCoordsToPixel

---

**Vector2f sf::RenderTarget::mapPixelToCoords (const Vector2i & point, const View & view) const**

Convert a point from target coordinates to world coordinates.

This function finds the 2D position that matches the given pixel of the render target. It does the inverse of what the graphics card does, to find the initial position of a rendered pixel.

Initially, both coordinate systems (world units and target pixels) match perfectly. But if you define a custom view or resize your render target, this assertion is not true anymore. A point located at (10, 50) in your render target may map to the point (150, 75) in your 2D world if the view is translated by (140, 25).

For render-windows, this function is typically used to find which point is located below the mouse cursor.

This version uses a custom view for calculations, see the other overload.
use the current view of the render target.

**Parameters**

- `point` Pixel to convert
- `view` The view to use for converting the point

**Returns**

The converted point, in "world" units

**See also**

- `mapCoordsToPixel`

---

**void sf::RenderTarget::popGLStates ( )**

Restore the previously saved OpenGL render states and matrices.

See the description of `pushGLStates` to get a detailed description of these functions.

**See also**

- `pushGLStates`

---

**void sf::RenderTarget::pushGLStates ( )**

Save the current OpenGL render states and matrices.

This function can be used when you mix SFML drawing and direct OpenGL rendering. Combined with `popGLStates`, it ensures that:

- SFML's internal states are not messed up by your OpenGL code
- your OpenGL states are not modified by a call to a SFML function
More specifically, it must be used around code that calls Draw functions.

```cpp
// OpenGL code here...
window.pushGLStates();
window.draw(...);
window.draw(...);
window.popGLStates();
// OpenGL code here...
```

Note that this function is quite expensive: it saves all the possible OpenGL states you don’t care about. Therefore it should be used wisely. It is probably best results will be achieved if you handle OpenGL states yourself (because really changed, and need to be saved and restored). Take a look at the so.

**See also**

`popGLStates`

### void sf::RenderTarget::resetGLStates ()

Reset the internal OpenGL states so that the target is ready for drawing.

This function can be used when you mix SFML drawing and direct OpenGL to use `pushGLStates/popGLStates`. It makes sure that all OpenGL states that subsequent `draw()` calls will work as expected.

**Example:**

```cpp
// OpenGL code here...
glPushAttrib(...);
window.resetGLStates();
window.draw(...);
window.draw(...);
glPopAttrib(...);
// OpenGL code here...
```
bool sf::RenderTexture::setActive ( bool active = true )

Activate or deactivate the render-texture for rendering.

This function makes the render-texture's context current for future OpenGL rendering operations (so you shouldn't care about it if you're not doing direct OpenGL stuff). Only one context can be current in a thread, so if you want to draw OpenGL geometry to another render target, forget to activate it again.

Parameters

active True to activate, false to deactivate

Returns

True if operation was successful, false otherwise

void sf::RenderTexture::setRepeated ( bool repeated )

Enable or disable texture repeating.

This function is similar to Texture::setRepeated. This parameter is disabled by default.

Parameters

repeated True to enable repeating, false to disable it

See also

isRepeated

void sf::RenderTexture::setSmooth ( bool smooth )

Enable or disable texture smoothing.
This function is similar to `Texture::setSmooth`. This parameter is disabled by default.

**Parameters**

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

**See also**

- `isSmooth`

---

**void sf::RenderTarget::setView (const View & view)**

Change the current active view.

The view is like a 2D camera, it controls which part of the 2D scene is visible in the render target. The new view will affect everything that is drawn, until another view is set. The render target keeps its own copy of the view object, so it is not necessary to keep the original one alive after calling this function. To restore the original view of the target, you can pass the result of this function.

**Parameters**

- **view** New view to use

**See also**

- `getView`, `getDefaultView`

---

The documentation for this class was generated from the following file:
- `RenderTexture.hpp`

---

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila. :: Documentation generated by doxygen.
sf::RenderWindow Class Reference

Window that can serve as a target for 2D drawing. More...

#include <RenderWindow.hpp>

Inheritance diagram for sf::RenderWindow:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RenderWindow ()</strong></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td><strong>RenderWindow (VideoMode mode, const String &amp;title, Uint32 style=Style::Default, const ContextSettings &amp;settings=ContextSettings())</strong></td>
<td>Construct a new window. More...</td>
</tr>
<tr>
<td><strong>RenderWindow (WindowHandle handle, const &amp;settings=ContextSettings())</strong></td>
<td>Construct the window from an existing control.</td>
</tr>
<tr>
<td><strong>virtual ~RenderWindow ()</strong></td>
<td>Destructor. More...</td>
</tr>
<tr>
<td><strong>virtual Vector2u getSize () const</strong></td>
<td>Get the size of the rendering region of the window.</td>
</tr>
<tr>
<td><strong>Image capture () const</strong></td>
<td>Copy the current contents of the window to an image.</td>
</tr>
<tr>
<td><strong>void create (VideoMode mode, const String &amp;title, const ContextSettings &amp;settings=ContextSettings())</strong></td>
<td>Create (or recreate) the window. More...</td>
</tr>
<tr>
<td><strong>void create (WindowHandle handle, const ContextSettings &amp;settings=ContextSettings())</strong></td>
<td>Create (or recreate) the window from an existing control.</td>
</tr>
<tr>
<td><strong>void close ()</strong></td>
<td>Close the window and destroy all the attached resources.</td>
</tr>
</tbody>
</table>
bool isOpen () const
Tell whether or not the window is open. More...

const ContextSettings & getSettings () const
Get the settings of the OpenGL context of the window.

bool pollEvent (Event &event)
Pop the event on top of the event queue, if any.

bool waitEvent (Event &event)
Wait for an event and return it. More...

Vector2i getPosition () const
Get the position of the window. More...

void setPosition (const Vector2i &position)
Change the position of the window on screen.

void setSize (const Vector2u &size)
Change the size of the rendering region of the window.

void setTitle (const String &title)
Change the title of the window. More...

void setIcon (unsigned int width, unsigned int height, const Uint8 *pixels)
Change the window's icon. More...

void setVisible (bool visible)
Show or hide the window. More...

void setVerticalSyncEnabled (bool enabled)
Enable or disable vertical synchronization. More...

void setMouseCursorVisible (bool visible)
Show or hide the mouse cursor. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setMouseCursorGrabbed (bool grabbed)</td>
<td>Grab or release the mouse cursor.</td>
</tr>
<tr>
<td>void setKeyRepeatEnabled (bool enabled)</td>
<td>Enable or disable automatic key-repeat.</td>
</tr>
<tr>
<td>void setFramerateLimit (unsigned int limit)</td>
<td>Limit the framerate to a maximum fixed frequency.</td>
</tr>
<tr>
<td>void setJoystickThreshold (float threshold)</td>
<td>Change the joystick threshold.</td>
</tr>
<tr>
<td>bool setActive (bool active=true) const</td>
<td>Activate or deactivate the window as the current target for OpenGL rendering.</td>
</tr>
<tr>
<td>void requestFocus ()</td>
<td>Request the current window to be made the active foreground window.</td>
</tr>
<tr>
<td>bool hasFocus () const</td>
<td>Check whether the window has the input focus.</td>
</tr>
<tr>
<td>void display ()</td>
<td>Display on screen what has been rendered to the window so far.</td>
</tr>
<tr>
<td>WindowHandle getSystemHandle () const</td>
<td>Get the OS-specific handle of the window.</td>
</tr>
<tr>
<td>void clear (const Color &amp;color=Color(0, 0, 0, 255))</td>
<td>Clear the entire target with a single color.</td>
</tr>
<tr>
<td>void setView (const View &amp;view)</td>
<td>Change the current active view.</td>
</tr>
</tbody>
</table>
const View & getView () const
Get the view currently in use in the render target.

const View & getDefaultView () const
Get the default view of the render target. More...

IntRect getViewport (const View &view) const
Get the viewport of a view, applied to this render target.

Vector2f mapPixelToCoords (const Vector2i &point) const
Convert a point from target coordinates to world view. More...

Vector2f mapPixelToCoords (const Vector2i &point, const Convert a point from target coordinates to world view.

Vector2i mapCoordsToPixel (const Vector2f &point) const
Convert a point from world coordinates to target view. More...

Vector2i mapCoordsToPixel (const Vector2f &point, const Convert a point from world coordinates to target view.

void draw (const Drawable &drawable, const RenderStates &states=RenderStates::Default)
Draw a drawable object to the render target. More...

void draw (const Vertex *vertices, std::size_t vertexCount, RenderStates &states=RenderStates::Default)
Draw primitives defined by an array of vertices.

void pushGLStates ()
Save the current OpenGL render states and matrices.

void popGLStates ()
Restore the previously saved OpenGL render states and matrices.
void resetGLStates ()
Reset the internal OpenGL states so that the target is ready for drawing.
More...
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual void onCreate ()</td>
<td>Function called after the window has been created. [More...]</td>
</tr>
<tr>
<td>virtual void onResize ()</td>
<td>Function called after the window has been resized. [More...]</td>
</tr>
<tr>
<td>void initialize ()</td>
<td>Performs the common initialization step after creation. [More.]</td>
</tr>
</tbody>
</table>
Detailed Description

Window that can serve as a target for 2D drawing.

sf::RenderWindow is the main class of the Graphics module.

It defines an OS window that can be painted using the other classes of the

sf::RenderWindow is derived from sf::Window, thus it inherits all its features: OpenGL rendering, etc. See the documentation of sf::Window for a more complete description, as well as code examples.

On top of that, sf::RenderWindow adds more features related to 2D drawing (see its base class sf::RenderTarget for more details). Here is a typical rendering and event loop using sf::RenderWindow:

```cpp
// Declare and create a new render-window
sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");

// Limit the framerate to 60 frames per second (this step is optional)
window.setFramerateLimit(60);

// The main loop - ends as soon as the window is closed
while (window.isOpen())
{
    // Event processing
    sf::Event event;
    while (window.pollEvent(event))
    {
        // Request for closing the window
        if (event.type == sf::Event::Closed)
            window.close();
    }

    // Clear the whole window before rendering a new frame
    window.clear();

    // Draw some graphical entities
    window.draw(sprite);
    window.draw(circle);
```
Like `sf::Window`, `sf::RenderWindow` is still able to render direct OpenGL calls together OpenGL calls and regular SFML drawing commands.

```cpp
// Create the render window
sf::RenderWindow window(sf::VideoMode(800, 600), "SFML OpenGL");

// Create a sprite and a text to display
sf::Sprite sprite;
sf::Text text;
...

// Perform OpenGL initializations
glMatrixMode(GL_PROJECTION);
...

// Start the rendering loop
while (window.isOpen())
{
    // Process events
    ...

    // Draw a background sprite
    window.pushGLStates();
    window.draw(sprite);
    window.popGLStates();

    // Draw a 3D object using OpenGL
    glBegin(GL_QUADS);
        glVertex3f(...);
        ...
    glEnd();

    // Draw text on top of the 3D object
    window.pushGLStates();
    window.draw(text);
    window.popGLStates();

    // Finally, display the rendered frame on screen
    window.display();
}
```

See also
- `sf::Window`, `sf::RenderTarget`, `sf::RenderTexture`, `sf::View`
Definition at line 44 of file RenderWindow.hpp.
Constructor & Destructor Documentation

sf::RenderWindow::RenderWindow ( )

Default constructor.

This constructor doesn't actually create the window, use the other constructor.

sf::RenderWindow::RenderWindow ( VideoMode mode, const String & title, Uint32 style, const ContextSettings & settings )

Construct a new window.

This constructor creates the window with the size and pixel depth define be passed to customize the look and behavior of the window (borders, title).

The fourth parameter is an optional structure specifying advanced OpenGL antialiasing, depth-buffer bits, etc. You shouldn't care about these parameters for regular usage of the graphics module.

Parameters

- **mode**: Video mode to use (defines the width, height and depth of window)
- **title**: Title of the window
- **style**: Window style, a bitwise OR combination of sf::Style enumerations
sf::RenderWindow::RenderWindow ( WindowHandle handle, const ContextSettings & settings )

Construct the window from an existing control.

Use this constructor if you want to create an SFML rendering area into an existing control.

The second parameter is an optional structure specifying advanced OpenGL context settings such as antialiasing, depth-buffer bits, etc. You shouldn't care about these parameters for a regular usage of the graphics module.

Parameters

- **handle** Platform-specific handle of the control (HWND on Windows, NSWindow on OS X)
- **settings** Additional settings for the underlying OpenGL context

virtual sf::RenderWindow::~RenderWindow ( )

Destructor.

Closes the window and frees all the resources attached to it.
Member Function Documentation

**Image** sf::RenderWindow::capture ( ) const

Copy the current contents of the window to an image.

**Deprecated:**

Use a sf::Texture and its sf::Texture::update(const Window&) function instead.

```cpp
sf::Vector2u windowSize = window.getSize();
sf::Texture texture;
texture.create(windowSize.x, windowSize.y);
texture.update(window);
sf::Image screenshot = texture.copyToImage();
```

This is a slow operation, whose main purpose is to make screenshots or update an image with the contents of the window and then use it for drawing

sf::Texture and its update(Window&) function. You can also draw things to a

sf::RenderTarget class.

**Returns**

Image containing the captured contents

---

**void** sf::RenderTarget::clear ( const Color & color = Color(0, 0, 0, 255) )

Clear the entire target with a single color.

This function is usually called once every frame, to clear the previous contents.
**Parameters**

- **color** Fill color to use to clear the render target

---

**void sf::Window::close ()**

Close the window and destroy all the attached resources.

After calling this function, the `sf::Window` instance remains valid and you can use it. All other functions such as `pollEvent()` or `display()` will still work (i.e., you don’t have to test `isOpen()` every time), and will have no effect on closed windows.

---

**void sf::Window::create (** VideoMode **mode, const String &** title, **Uint32** style = Style::Default, **const ContextSettings &** settings = ContextSettings **)***

Create (or recreate) the window.

If the window was already created, it closes it first. If `style` contains Style::Fullscreen, then the window will be created with a valid video mode.

The fourth parameter is an optional structure specifying advanced OpenGL context settings such as antialiasing, depth-buffer bits, etc.

**Parameters**

- **mode** Video mode to use (defines the width, height and depth of window)
- **title** Title of the window
- **style** Window style, a bitwise OR combination of `sf::Style` enumerators
- **settings** Additional settings for the underlying OpenGL context
void sf::Window::create (WindowHandle handle, const ContextSettings & settings = ContextSettings())

Create (or recreate) the window from an existing control.

Use this function if you want to create an OpenGL rendering area into an already existing control. If the window was already created, it closes it first.

The second parameter is an optional structure specifying advanced OpenGL context settings such as antialiasing, depth-buffer bits, etc.

Parameters

- **handle** Platform-specific handle of the control
- **settings** Additional settings for the underlying OpenGL context

void sf::Window::display ()

Display on screen what has been rendered to the window so far.

This function is typically called after all OpenGL rendering has been done to show it on screen.

void sf::RenderTarget::draw (const Drawable & drawable, const RenderStates & states = RenderStates())

Draw a drawable object to the render target.
Draw primitives defined by an array of vertices.

**Parameters**

- **vertices**
  - Pointer to the vertices
- **vertexCount**
  - Number of vertices in the array
- **type**
  - Type of primitives to draw
- **states**
  - Render states to use for drawing

Get the default view of the render target.

The default view has the initial size of the render target, and never changes after the target has been created.

**Returns**

- The default view of the render target

**See also**

- `setView`, `getView`
**Vector2i sf::Window::getPosition ( ) const**

Get the position of the window.

**Returns**

Position of the window, in pixels

**See also**

setPosition

**const ContextSettings& sf::Window::getSettings ( ) const**

Get the settings of the OpenGL context of the window.

Note that these settings may be different from what was passed to the function, if one or more settings were not supported. In this case, SFML chooses the closest match.

**Returns**

Structure containing the OpenGL context settings

**virtual Vector2u sf::RenderWindow::getSize ( ) const**

Get the size of the rendering region of the window.

The size doesn't include the titlebar and borders of the window.

**Returns**

Size in pixels

Implements sf::RenderTarget.
**WindowHandle**  sf::Window::getSystemHandle ( ) const

Get the OS-specific handle of the window.

The type of the returned handle is sf::WindowHandle, which is a typedef to the OS. 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.

**Returns**

System handle of the window

---

**const View** & sf::RenderTarget::getView ( ) const

Get the view currently in use in the render target.

**Returns**

The view object that is currently used

**See also**

setView, getDefaultView

---

**IntRect** sf::RenderTarget::getViewport ( const View & view ) const

Get the viewport of a view, applied to this render target.

The viewport is defined in the view as a ratio, this function simply applies the dimensions of the render target to calculate the pixels rectangle that the viewport targets.
### Parameters

**view** The view for which we want to compute the viewport

### Returns

Viewport rectangle, expressed in pixels

---

**bool sf::Window::hasFocus ( ) const**

Check whether the window has the input focus.

At any given time, only one window may have the input focus to receive or most mouse events.

### Returns

True if window has focus, false otherwise

### See also

`requestFocus`

---

**void sf::RenderTarget::initialize ( )**

Performs the common initialization step after creation.

The derived classes must call this function after the target is created and

---

**bool sf::Window::isOpen ( ) const**

Tell whether or not the window is open.
This function returns whether or not the window exists. Note that a hidden window is considered open (therefore this function would return true).

**Returns**

True if the window is open, false if it has been closed.

```
Vector2i sf::RenderTarget::mapCoordsToPixel ( const Vector2f & point )
```

Convert a point from world coordinates to target coordinates, using the current view.

This function is an overload of the `mapCoordsToPixel` function that implicitly uses the current view. It is equivalent to:

```
target.mapCoordsToPixel(point, target.getView());
```

**Parameters**

- `point` Point to convert

**Returns**

The converted point, in target coordinates (pixels)

**See also**

- `mapPixelToCoords`

```
Vector2i sf::RenderTarget::mapCoordsToPixel ( const Vector2f & point, const View & view )
```

Convert a point from world coordinates to target coordinates.

This function finds the pixel of the render target that matches the given...
through the same process as the graphics card, to compute the final pos

Initially, both coordinate systems (world units and target pixels) match
custom view or resize your render target, this assertion is not true anym
75) in your 2D world may map to the pixel (10, 50) of your render tar
g (140, 25).

This version uses a custom view for calculations, see the other overload
use the current view of the render target.

Parameters

- **point**  Point to convert
- **view**   The view to use for converting the point

Returns

The converted point, in target coordinates (pixels)

See also

- [mapPixelToCoords](sf::RenderTarget::mapPixelToCoords)

---

**Vector2f sf::RenderTarget::mapPixelToCoords (const Vector2i & point, View & view)**

Convert a point from target coordinates to world coordinates, using the c

This function is an overload of the mapPixelToCoords function that imp
equivalent to:

```cpp
target.mapPixelToCoords(point, target.getView());
```

Parameters

- **point**  Pixel to convert

Returns
The converted point, in "world" coordinates

See also
mapCoordsToPixel

```
Vector2f sf::RenderTarget::mapPixelToCoords ( const Vector2i & point, const View & view ) const
```

Convert a point from target coordinates to world coordinates.

This function finds the 2D position that matches the given pixel of the render target and does the inverse of what the graphics card does, to find the initial position.

Initially, both coordinate systems (world units and target pixels) match perfectly. But if you define a custom view or resize your render target, this assertion is not true anymore. For example, a point located at (10, 50) in your render target may map to the point (150, 75) in your 2D world coordinates (140, 25).

For render-windows, this function is typically used to find which point is located below the mouse cursor.

This version uses a custom view for calculations, see the other overload if you want to use the current view of the render target.

Parameters

- **point** Pixel to convert
- **view** The view to use for converting the point

Returns

The converted point, in "world" units

See also
**virtual void sf::RenderWindow::onCreate ( )**

Function called after the window has been created.

This function is called so that derived classes can perform their own specific initialization as soon as the window is created.

Reimplemented from sf::Window.

**virtual void sf::RenderWindow::onResize ( )**

Function called after the window has been resized.

This function is called so that derived classes can perform custom actions when the size of the window changes.

Reimplemented from sf::Window.

**bool sf::Window::pollEvent ( Event & event )**

Pop the event on top of the event queue, if any, and return it.

This function is not blocking: if there's no pending event then it will return false unmodified. Note that more than one event may be present in the event queue, so you should call this function in a loop to make sure that you process every pending event.

```cpp
sf::Event event;
while (window.pollEvent(event))
```
Parameters

**event** `Event` to be returned

**Returns**

True if an event was returned, or false if the event queue was empty

**See also**

`waitEvent`

---

**void sf::RenderTarget::popGLStates ( )**

Restore the previously saved OpenGL render states and matrices.

See the description of `pushGLStates` to get a detailed description of these functions.

**See also**

`pushGLStates`

---

**void sf::RenderTarget::pushGLStates ( )**

Save the current OpenGL render states and matrices.

This function can be used when you mix SFML drawing and direct OpenGL rendering. Combined with `popGLStates`, it ensures that:

- SFML’s internal states are not messed up by your OpenGL code
- your OpenGL states are not modified by a call to a SFML function
More specifically, it must be used around code that calls Draw functions.

```cpp
// OpenGL code here...
window.pushGLStates();
window.draw(...);
window.draw(...);
window.popGLStates();
// OpenGL code here...
```

Note that this function is quite expensive: it saves all the possible OpenGL states you don't care about. Therefore it should be used wisely. It is pr
best results will be achieved if you handle OpenGL states yourself (beca really changed, and need to be saved and restored). Take a look at the so.

**See also**

`popGLStates`

---

### void sf::Window::requestFocus ( )

Request the current window to be made the active foreground window.

At any given time, only one window may have the input focus to receive or mouse events. If a window requests focus, it only hints to the operatin focused. The operating system is free to deny the request. This is not to

**See also**

`hasFocus`

---

### void sf::RenderTarget::resetGLStates ( )

Reset the internal OpenGL states so that the target is ready for drawing.
This function can be used when you mix SFML drawing and direct OpenGL rendering, if you choose not to use pushGLStates/popGLStates. It makes sure that all OpenGL states needed by SFML are set, so subsequent `draw()` calls will work as expected.

Example:

```cpp
// OpenGL code here...
glPushAttrib(...);
window.resetGLStates();
window.draw(...);
window.draw(...);
glPopAttrib(...);
// OpenGL code here...
```

### bool sf::Window::setActive (bool active = true) const

Activate or deactivate the window as the current target for OpenGL rendering.

A window is active only on the current thread, if you want to make it active on another thread you have to deactivate it on the previous thread first if it was active. Only one window can be active on a thread at a time, thus the window previously active (if any) automatically gets deactivated with `requestFocus()`.

**Parameters**

- `active` True to activate, false to deactivate

**Returns**

- True if operation was successful, false otherwise

### void sf::Window::setFramerateLimit (unsigned int limit)

Limit the framerate to a maximum fixed frequency.

If a limit is set, the window will use a small delay after each call to `display()`.
frame lasted long enough to match the framerate limit. SFML will try to run
it can, but since it internally uses sf::sleep, whose precision depends on
may be a little unprecise as well (for example, you can get 65 FPS when

Parameters

| limit | Framerate limit, in frames per seconds (use 0 to disable limit) |

```cpp
void sf::Window::setIcon ( unsigned int width,
                          unsigned int height,
                          const Uint8 * pixels  )
```

Change the window's icon.

*pixels* must be an array of *width* x *height* pixels in 32-bits RGBA format.

The OS default icon is used by default.

Parameters

- **width** Icon’s width, in pixels
- **height** Icon’s height, in pixels
- **pixels** Pointer to the array of pixels in memory. The pixels are copied
source alive after calling this function.

See also

- setTitle

```cpp
void sf::Window::setJoystickThreshold ( float threshold )
```

Change the joystick threshold.
The joystick threshold is the value below which no JoystickMoved event will be generated. The threshold value is 0.1 by default.

**Parameters**

- **threshold** New threshold, in the range [0, 100]

---

```cpp
void sf::Window::setKeyRepeatEnabled ( bool enabled )
```

Enable or disable automatic key-repeat.

If key repeat is enabled, you will receive repeated KeyPressed events while it is disabled, you will only get a single event when the key is pressed.

Key repeat is enabled by default.

**Parameters**

- **enabled** True to enable, false to disable

---

```cpp
void sf::Window::setMouseCursorGrabbed ( bool grabbed )
```

Grab or release the mouse cursor.

If set, grabs the mouse cursor inside this window's client area so it may not be moved outside its bounds. Note that grabbing is only active while the window has focus.

**Parameters**

- **grabbed** True to enable, false to disable

---

```cpp
void sf::Window::setMouseCursorVisible ( bool visible )
```
Show or hide the mouse cursor.

The mouse cursor is visible by default.

**Parameters**

- `visible` True to show the mouse cursor, false to hide it

```cpp
void sf::Window::setPosition (const Vector2i & position)
```

Change the position of the window on screen.

This function only works for top-level windows (i.e. it will be ignored for windows created from the handle of a child window/control).

**Parameters**

- `position` New position, in pixels

**See also**

- `getPosition`

```cpp
void sf::Window::setSize (const Vector2u & size)
```

Change the size of the rendering region of the window.

**Parameters**

- `size` New size, in pixels

**See also**

- `getSize`
### void sf::Window::setTitle ( const String & title )

Change the title of the window.

**Parameters**
- **title** New title

**See also**
- setIcon

### void sf::Window::setVerticalSyncEnabled ( bool enabled )

Enable or disable vertical synchronization.

Activating vertical synchronization will limit the number of frames disp monitor. This can avoid some visual artifacts, and limit the framerate to across different computers).

Vertical synchronization is disabled by default.

**Parameters**
- **enabled** True to enable v-sync, false to deactivate it

### void sf::RenderTarget::setView ( const View & view )

Change the current active view.

The view is like a 2D camera, it controls which part of the 2D scene is visible in the render target. The new view will affect everything that is drawn, until...
target keeps its own copy of the view object, so it is not necessary to calling this function. To restore the original view of the target, you can pa to this function.

**Parameters**

- **view** New view to use

**See also**

getView, getDefaultView

---

**void sf::Window::setVisible ( bool visible )**

Show or hide the window.

The window is shown by default.

**Parameters**

- **visible** True to show the window, false to hide it

---

**bool sf::Window::waitEvent ( Event & event )**

Wait for an event and return it.

This function is blocking: if there's no pending event then it will wait unt function returns (and no error occurred), the `event` object is always valid is typically used when you have a thread that is dedicated to events handling thread sleep as long as no new event is received.

```cpp
sf::Event event;
if (window.waitEvent(event))
{
    // process event...
```
Parameters
  event Event to be returned

Returns
  False if any error occurred

See also
  pollEvent

The documentation for this class was generated from the following file:
- RenderWindow.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen
sf::Shader Class Reference
Graphics module

Shader class (vertex, geometry and fragment) More...

#include <Shader.hpp>

Inheritance diagram for sf::Shader:
## Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>CurrentTextureType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special type that can be passed to <code>setUniform()</code> and that represents the texture of the object being drawn.</td>
<td>More...</td>
</tr>
</tbody>
</table>
Public Types

<table>
<thead>
<tr>
<th>enum</th>
<th>Type { Vertex, Geometry, Fragment }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Types of shaders. More...</td>
</tr>
</tbody>
</table>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shader ()</strong>&lt;br&gt;Default constructor. More...&lt;br&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>~Shader ()</strong>&lt;br&gt;Destructor. More...&lt;br&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromFile (const std::string &amp;filename, Type type)</strong>&lt;br&gt;Load the vertex, geometry or fragment shader from a file. More...&lt;br&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromFile (const std::string &amp;vertexShaderFilename, const std::string &amp;fragmentShaderFilename)</strong>&lt;br&gt;Load both the vertex and fragment shaders from files. More...&lt;br&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromFile (const std::string &amp;vertexShaderFilename, const std::string &amp;geometryShaderFilename, const std::string &amp;fragmentShaderFilename)</strong>&lt;br&gt;Load the vertex, geometry and fragment shaders from files.</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromMemory (const std::string &amp;shader, Type type)</strong>&lt;br&gt;Load the vertex, geometry or fragment shader from a source code. More...&lt;br&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromMemory (const std::string &amp;vertexShader, const std::string &amp;fragmentShader)</strong>&lt;br&gt;Load both the vertex and fragment shaders from source code. More...&lt;br&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromMemory (const std::string &amp;vertexShader, const std::string &amp;geometryShader, const std::string &amp;fragmentShader)</strong>&lt;br&gt;Load the vertex, geometry and fragment shaders from source code. More...</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromStream (InputStream &amp;stream, Type type)</strong>&lt;br&gt;Load the vertex, geometry or fragment shader from a custom stream. More...&lt;br&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>bool loadFromStream (InputStream &amp;vertexShaderStream, InputStream &amp;fragmentShaderStream, InputStream &amp;geometryShaderStream)</strong>&lt;br&gt;Load the vertex, geometry and fragment shaders from custom streams. More...</td>
<td></td>
</tr>
</tbody>
</table>
Load both the vertex and fragment shaders from custom streams.

```cpp
bool loadFromStream (InputStream &vertexShaderStream, InputStream &geometryShaderStream, InputStream &fragmentShaderStream)
```

Load the vertex, geometry and fragment shaders from custom streams.

```cpp
void setUniform (const std::string &name, float x)
```

Specify value for float uniform. More...

```cpp
void setUniform (const std::string &name, const Glsl::Vec2 &vector)
```

Specify value for vec2 uniform. More...

```cpp
void setUniform (const std::string &name, const Glsl::Vec3 &vector)
```

Specify value for vec3 uniform. More...

```cpp
void setUniform (const std::string &name, const Glsl::Vec4 &vector)
```

Specify value for vec4 uniform. More...

```cpp
void setUniform (const std::string &name, int x)
```

Specify value for int uniform. More...

```cpp
void setUniform (const std::string &name, const Glsl::Ivec2 &vector)
```

Specify value for ivec2 uniform. More...

```cpp
void setUniform (const std::string &name, const Glsl::Ivec3 &vector)
```

Specify value for ivec3 uniform. More...

```cpp
void setUniform (const std::string &name, const Glsl::Ivec4 &vector)
```

Specify value for ivec4 uniform. More...

```cpp
void setUniform (const std::string &name, bool x)
```

Specify value for bool uniform. More...

```cpp
void setUniform (const std::string &name, const Glsl::Bvec2 &vector)
```

Specify value for bvec2 uniform. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setUniform (const std::string &amp;name, const Glsl::Bvec2 &amp;vector)</code></td>
<td>Specify value for <code>bvec2</code> uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniform (const std::string &amp;name, const Glsl::Bvec3 &amp;vector)</code></td>
<td>Specify value for <code>bvec3</code> uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniform (const std::string &amp;name, const Glsl::Bvec4 &amp;vector)</code></td>
<td>Specify value for <code>bvec4</code> uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniform (const std::string &amp;name, const Glsl::Mat3 &amp;matrix)</code></td>
<td>Specify value for <code>mat3</code> matrix. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniform (const std::string &amp;name, const Glsl::Mat4 &amp;matrix)</code></td>
<td>Specify value for <code>mat4</code> matrix. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniform (const std::string &amp;name, const Texture &amp;texture)</code></td>
<td>Specify a texture as <code>sampler2D</code> uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniform (const std::string &amp;name, CurrentTextureType)</code></td>
<td>Specify current texture as <code>sampler2D</code> uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniformArray (const std::string &amp;name, const float *scalarArray, std::size_t length)</code></td>
<td>Specify values for <code>float[]</code> array uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniformArray (const std::string &amp;name, const Glsl::Vec2 &amp;vector)</code></td>
<td>Specify values for <code>vec2[]</code> array uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniformArray (const std::string &amp;name, const Glsl::Vec3 &amp;vector)</code></td>
<td>Specify values for <code>vec3[]</code> array uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniformArray (const std::string &amp;name, const Glsl::Vec4 &amp;vector)</code></td>
<td>Specify values for <code>vec4[]</code> array uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setUniformArray (const std::string &amp;name, const Glsl::Mat3 &amp;matrix)</code></td>
<td>Specify values for <code>mat3[]</code> array uniform. More...</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>void setUniformArray (const std::string &amp;name, const Glsl::Mat4 &amp;array)</code></td>
<td>Specify values for <code>mat4[]</code> array uniform. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, float x)</code></td>
<td>Change a float parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, float x, float y)</code></td>
<td>Change a 2-components vector parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, float x, float y, float z)</code></td>
<td>Change a 3-components vector parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, float x, float y, float z, float w)</code></td>
<td>Change a 4-components vector parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, const Vector2f &amp;vector)</code></td>
<td>Change a 2-components vector parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, const Vector3f &amp;vector)</code></td>
<td>Change a 3-components vector parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, const Color &amp;color)</code></td>
<td>Change a color parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, const Transform &amp;transform)</code></td>
<td>Change a matrix parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, const Texture &amp;texture)</code></td>
<td>Change a texture parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>void setParameter (const std::string &amp;name, CurrentTextureType texture)</code></td>
<td>Change a texture parameter of the shader. More...</td>
<td></td>
</tr>
<tr>
<td><code>unsigned int getNativeHandle () const</code></td>
<td>Get the underlying OpenGL handle of the shader. More...</td>
<td></td>
</tr>
</tbody>
</table>
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>bind (const Shader *shader)</td>
<td>Bind a shader for rendering. <a href="#">More...</a></td>
</tr>
<tr>
<td>static bool</td>
<td>isAvailable ()</td>
<td>Tell whether or not the system supports shaders. <a href="#">More...</a></td>
</tr>
<tr>
<td>static bool</td>
<td>isGeometryAvailable ()</td>
<td>Tell whether or not the system supports geometry shaders. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Static Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>CurrentTextureType</td>
<td>Represents the texture of the object being drawn</td>
</tr>
</tbody>
</table>
Static Private Member Functions

static void ensureGLContext ()
Empty function for ABI compatibility, use acquireTransientCor
Detailed Description

**Shader class (vertex, geometry 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 three kinds of shaders:

- Vertex shaders, that process vertices
- Geometry shaders, that process primitives
- Fragment (pixel) shaders, that process pixels

A `sf::Shader` can be composed of either a vertex shader alone, a geometry shader alone, or any combination of them. (see the variants of the load function)

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

Like any C/C++ program, a GLSL shader has its own variables called *uniforms*:

- **scalars**: `float`, `int`, `bool`
- **vectors** (2, 3 or 4 components)
- **matrices** (3x3 or 4x4)
- **samplers** (textures)

Some SFML-specific types can be converted:

- `sf::Color` as a 4D vector (`vec4`)
- **sf::Transform** as matrices (**mat3** or **mat4**)

Every uniform variable in a shader can be set through one of the `setUniform()` overloads. For example, if you have a shader with the following uniforms:

```
uniform float offset;
uniform vec3 point;
uniform vec4 color;
uniform mat4 matrix;
uniform sampler2D overlay;
uniform sampler2D current;
```

You can set their values from C++ code as follows, using the types defined in the shader.

```cpp
shader.setUniform("offset", 2.f);
shader.setUniform("point", sf::Vector3f(0.5f, 0.8f, 0.3f));
shader.setUniform("color", sf::Glsl::Vec4(color));  // color is a sf::Color
shader.setUniform("matrix", sf::Glsl::Mat4(transform));  // transform
shader.setUniform("overlay", texture);  // texture
shader.setUniform("current", sf::Shader::CurrentTexture);
```

The old `setParameter()` overloads are deprecated and will be removed in a future version. You should use their `setUniform()` equivalents instead.

The special `sf::Shader::CurrentTexture` argument maps the given `sampler2D` the object being drawn (which cannot be known in advance).

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

```cpp
window.draw(sprite, &shader);
```

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

```cpp
sf::RenderStates states;
states.shader = &shader;
window.draw(sprite, states);
```
In the code above we pass a pointer to the shader, because it may be null.

Shaders can be used on any drawable, but some combinations are not interesting. For example, using a vertex shader on a `sf::Sprite` 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 texture of the text is not the actual text that you see on screen, it is characters of the font in an arbitrary order; thus, texture lookups on pixel may not give you the expected result.

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

- draw everything to a `sf::RenderTexture`, then draw it to the main target
- draw everything directly to the main target, then use `sf::Texture::update(Window&)` 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` that can be used as a raw OpenGL texture, `sf::Shader` can also be used directly as a raw shader for custom OpenGL geometry.

```cpp
sf::Shader::bind(&shader);
... render OpenGL geometry ...
sf::Shader::bind(NULL);
```

**See also**

`sf::Glsl`

Definition at line 52 of file `Shader.hpp`. 
## Member Enumeration Documentation

### `enum sf::Shader::Type`

Types of shaders.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex</td>
<td>Vertex shader</td>
</tr>
<tr>
<td>Geometry</td>
<td>Geometry shader.</td>
</tr>
<tr>
<td>Fragment</td>
<td>Fragment (pixel) shader.</td>
</tr>
</tbody>
</table>

Definition at line 60 of file `Shader.hpp`. 
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>sf::Shader::Shader ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default constructor.</td>
</tr>
<tr>
<td>This constructor creates an invalid shader.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf::Shader::~Shader ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destructor.</td>
</tr>
</tbody>
</table>

**Member Function Documentation**

### static void sf::Shader::bind ( const Shader * shader )

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` with OpenGL code.

```cpp
sf::Shader s1, s2;
...
sf::Shader::bind(&s1);
// draw OpenGL stuff that use s1...
sf::Shader::bind(&s2);
// draw OpenGL stuff that use s2...
sf::Shader::bind(NULL);
// draw OpenGL stuff that use no shader...
```

**Parameters**

- **shader** `Shader` to bind, can be null to use no shader

### unsigned int sf::Shader::getNativeHandle ( ) const

Get the underlying OpenGL handle of the shader.

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

**Returns**

- OpenGL handle of the shader or 0 if not yet loaded
**static bool sf::Shader::isAvailable ( )**

Tell whether or not the system supports shaders.

This function should always be called before using the shader features. Any attempt to use `sf::Shader` will fail.

**Returns**

- True if shaders are supported, false otherwise

**static bool sf::Shader::isGeometryAvailable ( )**

Tell whether or not the system supports geometry shaders.

This function should always be called before using the geometry shader features. Any attempt to use `sf::Shader` geometry shader features will fail.

This function can only return true if `isAvailable()` would also return true, since shaders in general have to be supported in order for geometry shaders to be supported as well.

**Note:** The first call to this function, whether by your code or SFML will result in a context switch.

**Returns**

- True if geometry shaders are supported, false otherwise

**bool sf::Shader::loadFromFile ( const std::string & filename, Type type )**
Load the vertex, geometry or fragment shader from a file.

This function loads a single shader, vertex, geometry or fragment, identified by the second argument. The source must be a text file containing a valid shader in GLSL 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, geometry or fragment shader file to load
- **type** Type of shader (vertex, geometry or fragment)

Returns

True if loading succeeded, false if it failed

See also

- `loadFromMemory`, `loadFromStream`

```cpp
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
Returns
True if loading succeeded, false if it failed

See also
loadFromMemory, loadFromStream

bool sf::Shader::loadFromFile ( const std::string & vertexShaderFilename, const std::string & geometryShaderFilename, const std::string & fragmentShaderFilename )

Load the vertex, geometry and fragment shaders from files.

This function loads the vertex, geometry and 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
geometryShaderFilename Path of the geometry shader file to load
fragmentShaderFilename Path of the fragment shader file to load

Returns
True if loading succeeded, false if it failed

See also
loadFromMemory, loadFromStream

bool sf::Shader::loadFromMemory ( const std::string & shader, Type type )
Load the vertex, geometry or fragment shader from a source code in memory.

This function loads a single shader, vertex, geometry 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 before writing your own shaders.

**Parameters**

- **shader** String containing the source code of the shader
- **type** Type of shader (vertex, geometry or fragment)

**Returns**

True if loading succeeded, false if it failed

**See also**

loadFromFile, loadFromStream

```cpp
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 is left empty (the valid shader is unloaded). The sources must be valid shaders in a C-like language dedicated to OpenGL shaders; you'll probably need to read a good documentation before writing your own shaders.

**Parameters**

- **vertexShader** String containing the source code of the vertex shader
- **fragmentShader** String containing the source code of the fragment shader

```cpp
bool sf::Shader::loadFromMemory ( const std::string & vertexShader,
const std::string & fragmentShader )
```
**fragmentShader** String containing the source code of the fragment

**Returns**
True if loading succeeded, false if it failed

**See also**
loadFromFile, loadFromStream

---

```cpp
bool sf::Shader::loadFromMemory ( const std::string & vertexShader,
                                   const std::string & geometryShader,
                                   const std::string & fragmentShader)
```

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

This function loads the vertex, geometry and fragment shaders. If one of them fails to load, the shader is left empty (the valid shader is unloaded). The sources must be valid shader codes written in 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 containing the source code of the vertex shader
- **geometryShader** String containing the source code of the geometry shader
- **fragmentShader** String containing the source code of the fragment shader

**Returns**
True if loading succeeded, false if it failed

**See also**
loadFromFile, loadFromStream
Load the vertex, geometry or fragment shader from a custom stream.

This function loads a single shader, vertex, geometry 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, geometry or fragment)

**Returns**

True if loading succeeded, false if it failed

**See also**

loadFromFile, loadFromMemory

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.
Load the vertex, geometry and fragment shaders from custom streams.

This function loads the vertex, geometry and 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 GLSL codes. 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
- `geometryShaderStream` Source stream to read the geometry shader from
- `fragmentShaderStream` Source stream to read the fragment shader from

**Returns**

True if loading succeeded, false if it failed

**See also**

`loadFromFile`, `loadFromMemory`
void sf::Shader::setParameter ( const std::string & name, float x )

Change a float parameter of the shader.

**Deprecated:**
Use setUniform(const std::string&, float) instead.

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

Change a 2-components vector parameter of the shader.

**Deprecated:**
Use setUniform(const std::string&, const Glsl::Vec2&) instead.

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

Change a 3-components vector parameter of the shader.

**Deprecated:**
Use setUniform(const std::string&, const Glsl::Vec3&) instead.
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.

**Deprecated:**
Use setUniform(const std::string&, const Glsl::Vec4&) instead.

void sf::Shader::setParameter ( const std::string & name, 
           const Vector2f & vector )

Change a 2-components vector parameter of the shader.

**Deprecated:**
Use setUniform(const std::string&, const Glsl::Vec2&) instead.

void sf::Shader::setParameter ( const std::string & name, 
           const Vector3f & vector )

Change a 3-components vector parameter of the shader.

**Deprecated:**
Use `setUniform(const std::string&, const Glsl::Vec3&)` instead.

```cpp
void sf::Shader::setParameter ( const std::string & name, const Color & color )
```

Change a color parameter of the shader.

**Deprecated:**
Use `setUniform(const std::string&, const Glsl::Vec4&)` instead.

```cpp
void sf::Shader::setParameter ( const std::string & name, const Transform & transform )
```

Change a matrix parameter of the shader.

**Deprecated:**
Use `setUniform(const std::string&, const Glsl::Mat4&)` instead.

```cpp
void sf::Shader::setParameter ( const std::string & name, const Texture & texture )
```

Change a texture parameter of the shader.

**Deprecated:**
Use `setUniform(const std::string&, const Texture&)` instead.
void sf::Shader::setParameter ( const std::string & name, CurrentTextureType )

Change a texture parameter of the shader.

**Deprecated:**
Use `setUniform(const std::string&, CurrentTextureType)` instead.

void sf::Shader::setUniform ( const std::string & name, float x )

Specify value for `float` uniform.

**Parameters**
- **name** Name of the uniform variable in GLSL
- **x** Value of the float scalar

void sf::Shader::setUniform ( const std::string & name, const Glsl::Vec2 & vector )

Specify value for `vec2` uniform.

**Parameters**
- **name** Name of the uniform variable in GLSL
- **vector** Value of the `vec2` vector
**void sf::Shader::setUniform** ( const std::string & name,  
    const Glsl::Vec3 & vector  
  )

Specify value for vec3 uniform.

**Parameters**

- **name**  Name of the uniform variable in GLSL
- **vector**  Value of the vec3 vector

**void sf::Shader::setUniform** ( const std::string & name,  
    const Glsl::Vec4 & vector  
  )

Specify value for vec4 uniform.

This overload can also be called with sf::Color objects that are converted to vec4 with normalized components before being passed to the shader. Therefore, they are converted from range [0 .. 255] to range [0 .. 1]. For example, `sf::Color(255, 127, 0, 255)` will be transformed to a vec4(1.0, 0.5, 0.0, 1.0).

**Parameters**

- **name**  Name of the uniform variable in GLSL
- **vector**  Value of the vec4 vector

**void sf::Shader::setUniform** ( const std::string & name,  
    int x  
  )

...
Specify value for \texttt{int} uniform.

**Parameters**
- \texttt{name} Name of the uniform variable in GLSL
- \texttt{x} Value of the int scalar

```cpp
void sf::Shader::setUniform ( const std::string & name, const Glsl::Ivec2 & vector )
```

Specify value for \texttt{ivec2} uniform.

**Parameters**
- \texttt{name} Name of the uniform variable in GLSL
- \texttt{vector} Value of the ivec2 vector

```cpp
void sf::Shader::setUniform ( const std::string & name, const Glsl::Ivec3 & vector )
```

Specify value for \texttt{ivec3} uniform.

**Parameters**
- \texttt{name} Name of the uniform variable in GLSL
- \texttt{vector} Value of the ivec3 vector

```cpp
void sf::Shader::setUniform ( const std::string & name, const Glsl::Ivec4 & vector )
```
Specify value for ivec4 uniform.

This overload can also be called with sf::Color objects that are converted to sf::Color instance. For example, sf::Color(255, 127, 0, 255) is mapped to ivec4(255, 127, 0, 255).

**Parameters**

- `name` Name of the uniform variable in GLSL
- `vector` Value of the ivec4 vector

```cpp
void sf::Shader::setUniform ( const std::string & name, bool x )
```

Specify value for bool uniform.

**Parameters**

- `name` Name of the uniform variable in GLSL
- `x` Value of the bool scalar

```cpp
void sf::Shader::setUniform ( const std::string & name, const Glsl::Bvec2 & vector )
```

Specify value for bvec2 uniform.

**Parameters**
**name**  Name of the uniform variable in GLSL

**vector**  Value of the bvec2 vector

```c++
void sf::Shader::setUniform ( const std::string & name,
                               const Glsl::Bvec3 & vector
                           )
```

Specify value for bvec3 uniform.

**Parameters**

- **name**  Name of the uniform variable in GLSL
- **vector**  Value of the bvec3 vector

```c++
void sf::Shader::setUniform ( const std::string & name,
                               const Glsl::Bvec4 & vector
                           )
```

Specify value for bvec4 uniform.

**Parameters**

- **name**  Name of the uniform variable in GLSL
- **vector**  Value of the bvec4 vector

```c++
void sf::Shader::setUniform ( const std::string & name,
                               const Glsl::Mat3 & matrix
                           )
```

Specify value for mat3 matrix.
Parameters

- **name**  Name of the uniform variable in GLSL
- **matrix** Value of the mat3 matrix

```cpp
void sf::Shader::setUniform ( const std::string & name,
                             const Glsl::Mat4 & matrix
                          )
```

Specify value for mat4 matrix.

Parameters

- **name**  Name of the uniform variable in GLSL
- **matrix** Value of the mat4 matrix

```cpp
void sf::Shader::setUniform ( const std::string & name,
                             const Texture & texture
                          )
```

Specify a texture as sampler2D uniform.

*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:

```cpp
uniform sampler2D the_texture; // this is the variable in the shader

sf::Texture texture;
...
shader.setUniform("the_texture", texture);
```

It is important to note that *texture* must remain alive as long as the s...
To use the texture of the object being drawn, which cannot be known in advance, you can pass the special value `sf::Shader::CurrentTexture`:

```cpp
shader.setUniform("the_texture", sf::Shader::CurrentTexture);
```

**Parameters**
- **name** Name of the texture in the shader
- **texture** Texture to assign

---

```cpp
void sf::Shader::setUniform ( const std::string & name,
                              CurrentTextureType
                           )
```

Specify current texture as `sampler2D` uniform.

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` parameter in the shader must be a 2D texture (`sampler2D` GLSL type).

Example:

```cpp
uniform sampler2D current; // this is the variable in the shader
shader.setUniform("current", sf::Shader::CurrentTexture);
```

**Parameters**
- **name** Name of the texture in the shader

---

```cpp
void sf::Shader::setUniformArray ( const std::string & name,
                                    const float * scalarArray,
                                    std::vector<float> vectorArray,
                                    std::array<float, 4> arrayArray
                                 )
```

---

```cpp
void sf::Shader::setUniformMatrix ( const std::string & name,
                                     const MatrixType& matrix
                                  )
```

---

```cpp
void sf::Shader::setUniformMatrix4x4 ( const std::string & name,
                                       const Matrix4x4Type& matrix
                                    )
```

---

```cpp
void sf::Shader::setUniformMatrix4x4Array ( const std::string & name,
                                             const Matrix4x4TypeArray& matrixArray
                                         )
```
Specify values for float[] array uniform.

Parameters

- **name**: Name of the uniform variable in GLSL
- **scalarArray**: pointer to array of float values
- **length**: Number of elements in the array

```cpp
define sf::Shader::setUniformArray(
    const std::string &name,
    const Glsl::Vec2 *vectorArray,
    std::size_t length
)
```

Specify values for vec2[] array uniform.

Parameters

- **name**: Name of the uniform variable in GLSL
- **vectorArray**: pointer to array of vec2 values
- **length**: Number of elements in the array

```cpp
define sf::Shader::setUniformArray(
    const std::string &name,
    const Glsl::Vec3 *vectorArray,
    std::size_t length
)
```

Specify values for vec3[] array uniform.

Parameters

- **name**: Name of the uniform variable in GLSL
- **vectorArray**: pointer to array of vec3 values
- **length**: Number of elements in the array

```cpp
define sf::Shader::setUniformArray(
    const std::string &name,
    const Glsl::Vec3 *vectorArray,
    std::size_t length
)
```
void sf::Shader::setUniformLocationWithMatrixArray(const std::string & name, const Glsl::Mat3 * matrixArray, std::size_t length)

Specify values for mat3[] array uniform.

Parameters
- name Name of the uniform variable in GLSL
- matrixArray pointer to array of mat3 values
- length Number of elements in the array
```cpp
const Glsl::Mat4 * matrixArray,
std::size_t length )
```

Specify values for `mat4[]` array uniform.

**Parameters**

- **name**  
  Name of the uniform variable in GLSL
- **matrixArray**  
  pointer to array of `mat4` values
- **length**  
  Number of elements in the array
Member Data Documentation

**CurrentTextureType** sf::Shader::CurrentTexture

Represents the texture of the object being drawn.

**See also**

```cpp
setUniform(const std::string&, CurrentTextureType)
```

Definition at line 82 of file Shader.hpp.

The documentation for this class was generated from the following file:
- Shader.hpp
sf::Shader::CurrentTextureType Struct Reference

Special type that can be passed to `setUniform()`, and that represents the texture being drawn.

More...

```cpp
#include <Shader.hpp>
```
Detailed Description

Special type that can be passed to `setUniform()`, and that represents the texture of the object being drawn.

**See also**

`setUniform(const std::string& current_texture_type)`

Definition at line 74 of file `Shader.hpp`.

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

- `Shader.hpp`
sf::Shape Class Reference

Base class for textured shapes with outline. More...

#include <Shape.hpp>

Inheritance diagram for sf::Shape:
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~Shape ()</td>
<td></td>
<td>Virtual destructor.</td>
</tr>
<tr>
<td>void setTexture (const Texture *texture, bool resetRect=false)</td>
<td></td>
<td>Change the source texture of the shape.</td>
</tr>
<tr>
<td>void setTextureRect (const IntRect &amp;rect)</td>
<td></td>
<td>Set the sub-rectangle of the texture that the shape will display.</td>
</tr>
<tr>
<td>void setFillColor (const Color &amp;color)</td>
<td></td>
<td>Set the fill color of the shape.</td>
</tr>
<tr>
<td>void setOutlineColor (const Color &amp;color)</td>
<td></td>
<td>Set the outline color of the shape.</td>
</tr>
<tr>
<td>void setOutlineThickness (float thickness)</td>
<td></td>
<td>Set the thickness of the shape's outline.</td>
</tr>
<tr>
<td>const Texture * getTexture () const</td>
<td></td>
<td>Get the source texture of the shape.</td>
</tr>
<tr>
<td>const IntRect &amp; getTextureRect () const</td>
<td></td>
<td>Get the sub-rectangle of the texture displayed by the shape.</td>
</tr>
<tr>
<td>const Color &amp; getFillColor () const</td>
<td></td>
<td>Get the fill color of the shape.</td>
</tr>
<tr>
<td>const Color &amp; getOutlineColor () const</td>
<td></td>
<td>Get the outline color of the shape.</td>
</tr>
<tr>
<td>float getOutlineThickness () const</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Get the outline thickness of the shape. More...

<table>
<thead>
<tr>
<th>virtual std::size_t</th>
<th>getPointCount () const =0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get the total number of points of the shape. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual Vector2f</th>
<th>getPoint (std::size_t index) const =0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get a point of the shape. More...</td>
</tr>
</tbody>
</table>

**FloatRect** getLocalBounds () const
Get the local bounding rectangle of the entity. More..

**FloatRect** getGlobalBounds () const
Get the global (non-minimal) bounding rectangle of the entity.

**void** setPosition (float x, float y)
set the position of the object More...

**void** setPosition (const Vector2f &position)
set the position of the object More...

**void** setRotation (float angle)
set the orientation of the object More...

**void** setScale (float factorX, float factorY)
set the scale factors of the object More...

**void** setScale (const Vector2f &factors)
set the scale factors of the object More...

**void** setOrigin (float x, float y)
set the local origin of the object More...

**void** setOrigin (const Vector2f &origin)
set the local origin of the object More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const Vector2f &amp; getPosition () const</code></td>
<td>get the position of the object</td>
<td></td>
</tr>
<tr>
<td><code>float getRotation () const</code></td>
<td>get the orientation of the object</td>
<td></td>
</tr>
<tr>
<td><code>const Vector2f &amp; getScale () const</code></td>
<td>get the current scale of the object</td>
<td></td>
</tr>
<tr>
<td><code>const Vector2f &amp; getOrigin () const</code></td>
<td>get the local origin of the object</td>
<td></td>
</tr>
<tr>
<td><code>void move (float offsetX, float offsetY)</code></td>
<td>Move the object by a given offset.</td>
<td>More...</td>
</tr>
<tr>
<td><code>void move (const Vector2f &amp;offset)</code></td>
<td>Move the object by a given offset.</td>
<td>More...</td>
</tr>
<tr>
<td><code>void rotate (float angle)</code></td>
<td>Rotate the object</td>
<td>More...</td>
</tr>
<tr>
<td><code>void scale (float factorX, float factorY)</code></td>
<td>Scale the object</td>
<td>More...</td>
</tr>
<tr>
<td><code>void scale (const Vector2f &amp;factor)</code></td>
<td>Scale the object</td>
<td>More...</td>
</tr>
<tr>
<td><code>const Transform &amp; getTransform () const</code></td>
<td>get the combined transform of the object</td>
<td>More...</td>
</tr>
<tr>
<td><code>const Transform &amp; getInverseTransform () const</code></td>
<td>get the inverse of the combined transform of the object</td>
<td>More...</td>
</tr>
</tbody>
</table>
# Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shape ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void update ()</strong></td>
<td>Recompute the internal geometry of the shape. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Base class for textured shapes with outline.

sf::Shape is a drawable class that allows to define and display a custom c

It's only an abstract base, it needs to be specialized for concrete types of
polygon, star, ...).

In addition to the attributes provided by the specialized shape classes, a
attributes:

- a texture
- a texture rectangle
- a fill color
- an outline color
- an outline thickness

Each feature is optional, and can be disabled easily:

- the texture can be null
- the fill/outline colors can be sf::Color::Transparent
- the outline thickness can be zero

You can write your own derived shape class, there are only two virtual fun

- getPointCount must return the number of points of the shape
- getPoint must return the points of the shape

See also
sf::RectangleShape, sf::CircleShape, sf::ConvexShape, sf::Transform

Definition at line 44 of file Shape.hpp.
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual sf::Shape::~Shape ( )</td>
<td>Virtual destructor.</td>
</tr>
<tr>
<td>sf::Shape::Shape ( )</td>
<td>Default constructor.</td>
</tr>
</tbody>
</table>
**Member Function Documentation**

---

**const Color& sf::Shape::getFillColor ( ) const**

Get the fill color of the shape.

**Returns**

Fill color of the shape

**See also**

setFillColor

---

**FloatRect sf::Shape::getGlobalBounds ( ) const**

Get the global (non-minimal) bounding rectangle of the entity.

The returned rectangle is in global coordinates, which means it takes into account transformations (translation, rotation, scale, ...) that are applied to the entity. This function returns the bounds of the shape in the global 2D world's coordinate system.

This function does not necessarily return the *minimal* bounding rectangle. The returned rectangle covers all the vertices (but possibly more). This allows for a fast approximation as a first check; you may want to use more precise checks on top of that.

**Returns**

Global bounding rectangle of the entity
**const Transform**& sf::Transformable::getInverseTransform ( ) const

get the inverse of the combined transform of the object

**Returns**
Inverse of the combined transformations applied to the object

**See also**
getTransform

---

**FloatRect** sf::Shape::getLocalBounds ( ) const

Get the local bounding rectangle of the entity.

The returned rectangle is in local coordinates, which means that (translation, rotation, scale, ...) that are applied to the entity. In other bounds of the entity in the entity's coordinate system.

**Returns**
Local bounding rectangle of the entity

---

**const Vector2f**& sf::Transformable::getOrigin ( ) const

get the local origin of the object

**Returns**
Current origin

**See also**
setOrigin
**const Color**& sf::Shape::getOutlineColor ( ) const

Get the outline color of the shape.

**Returns**

Outline color of the shape

**See also**

setOutlineColor

---

**float sf::Shape::getOutlineThickness ( ) const**

Get the outline thickness of the shape.

**Returns**

Outline thickness of the shape

**See also**

setOutlineThickness

---

**virtual Vector2f sf::Shape::getPoint ( std::size_t index ) const**

Get a point of the shape.

The returned point is in local coordinates, that is, the shape's transforms taken into account. The result is undefined if `index` is out of the valid range.

**Parameters**

`index` Index of the point to get, in range `[0 .. getPointCount() - 1]`
Returns

index-th point of the shape

See also

gPointCount

Implemented in sf::ConvexShape, sf::CircleShape, and sf::RectangleShape

virtual std::size_t sf::Shape::getPointCount ( ) const

Get the total number of points of the shape.

Returns

Number of points of the shape

See also

gPoint

Implemented in sf::CircleShape, sf::RectangleShape, and sf::ConvexShape

const Vector2f& sf::Transformable::getPosition ( ) const

get the position of the object

Returns

Current position

See also

setPosition
float sf::Transformable::getRotation ( ) const

get the orientation of the object

The rotation is always in the range [0, 360].

Returns
  Current rotation, in degrees

See also
  setRotation

const Vector2f& sf::Transformable::getScale ( ) const

get the current scale of the object

Returns
  Current scale factors

See also
  setScale

const Texture* sf::Shape::getTexture ( ) const

Get the source texture of the shape.

If the shape has no source texture, a NULL pointer is returned. This means that you can't modify the texture when you retrieve it with this function.

Returns
  Pointer to the shape's texture
See also

`setTexture`

`const IntRect& sf::Shape::getTextureRect ( ) const`

Get the sub-rectangle of the texture displayed by the shape.

**Returns**

Texture rectangle of the shape

See also

`setTextureRect`

`const Transform& sf::Transformable::getTransform ( ) const`

get the combined transform of the object

**Returns**

Transform combining the position/rotation(scale/origin) of the object

See also

`getInverseTransform`

`void sf::Transformable::move ( float offsetX, float offsetY )`

Move the object by a given offset.
This function adds to the current position of the object, unlike `setPosition` which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f pos = object.getPosition();
object.setPosition(pos.x + offsetX, pos.y + offsetY);
```

**Parameters**
- `offsetX` X offset
- `offsetY` Y offset

**See also**
- `setPosition`

---

```cpp
void sf::Transformable::move ( const Vector2f & offset )
```

Move the object by a given offset.

This function adds to the current position of the object, unlike `setPosition` which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setPosition(object.getPosition() + offset);
```

**Parameters**
- `offset` Offset

**See also**
- `setPosition`

---

```cpp
void sf::Transformable::rotate ( float angle )
```

Rotate the object.
This function adds to the current rotation of the object, unlike setRotation which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setRotation(object.getRotation() + angle);
```

**Parameters**

- **angle** Angle of rotation, in degrees

---

```cpp
void sf::Transformable::scale(float factorX, float factorY)
```

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factorX, scale.y * factorY);
```

**Parameters**

- **factorX** Horizontal scale factor
- **factorY** Vertical scale factor

**See also**

- `setScale`

---

```cpp
void sf::Transformable::scale(const Vector2f & factor)
```

Scale the object.
This function multiplies the current scale of the object, unlike `setScale` which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factor.x, scale.y * factor.y);
```

**Parameters**
- `factor` Scale factors

**See also**
- `setScale`

---

**void sf::Shape::setFillColor ( const Color & color )**

Set the fill color of the shape.

This color is modulated (multiplied) with the shape's texture if any. It can be used to colorize the shape, or change its global opacity. You can use `sf::Color::Transparent` to make the inside of the shape transparent, and have the outline alone. By default, the shape's fill color is opaque white.

**Parameters**
- `color` New color of the shape

**See also**
- `getFillColor`, `setOutlineColor`

---

**void sf::Transformable::setOrigin ( float x, float y )**

set the local origin of the object
The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object, and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

**Parameters**

- **x** X coordinate of the new origin
- **y** Y coordinate of the new origin

**See also**

- getOrigin

---

```cpp
void sf::Transformable::setOrigin ( const Vector2f & origin )
```

Set the local origin of the object.

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object, and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

**Parameters**

- **origin** New origin

**See also**

- getOrigin

---

```cpp
void sf::Shape::setOutlineColor ( const Color & color )
```

Set the outline color of the shape.

By default, the shape's outline color is opaque white.
**void sf::Shape::setOutlineThickness ( float thickness )**

Set the thickness of the shape's outline.

Note that negative values are allowed (so that the outline expands toward the center) using zero disables the outline. By default, the outline thickness is 0.

**Parameters**
- **thickness** New outline thickness

**See also**
- `getOutlineThickness`

**void sf::Transformable::setPosition ( float x, float y )**

set the position of the object

This function completely overwrites the previous position. See the move function to apply an offset based on the previous position instead. The default position of a transformable is (0, 0).

**Parameters**
- **x** X coordinate of the new position
- **y** Y coordinate of the new position

**Parameters**
- **color** New outline color of the shape

**See also**
- `getOutlineColor, setFillColor`
**void sf::Transformable::setPosition ( const Vector2f & position )**

set the position of the object

This function completely overwrites the previous position. See the move function to apply an offset based on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**

- **position** New position

**See also**

move, getPosition

---

**void sf::Transformable::setRotation ( float angle )**

set the orientation of the object

This function completely overwrites the previous rotation. See the rotate function to add an angle based on the previous rotation instead. The default rotation of a transformable object is 0.

**Parameters**

- **angle** New rotation, in degrees

**See also**

rotate, getRotation
void sf::Transformable::setScale ( float \texttt{factorX},
                                      float \texttt{factorY}
                                  )

set the scale factors of the object

This function completely overwrites the previous scale. See the scale \texttt{factorX} \texttt{factorY} the previous scale instead. The default scale of a transformable object is

\textbf{Parameters}
\begin{itemize}
  \item \texttt{factorX} New horizontal scale factor
  \item \texttt{factorY} New vertical scale factor
\end{itemize}

\textbf{See also}
\begin{itemize}
  \item \texttt{scale}, \texttt{getScale}
\end{itemize}

void sf::Transformable::setScale ( const \texttt{Vector2f} & \texttt{factors} )

set the scale factors of the object

This function completely overwrites the previous scale. See the scale \texttt{factors} the previous scale instead. The default scale of a transformable object is

\textbf{Parameters}
\begin{itemize}
  \item \texttt{factors} New scale factors
\end{itemize}

\textbf{See also}
\begin{itemize}
  \item \texttt{scale}, \texttt{getScale}
\end{itemize}

void sf::Shape::setTexture ( const \texttt{Texture} * \texttt{texture},
                                      \texttt{resetRect} = \texttt{false}
                                  )
Change the source texture of the shape.

The `texture` argument refers to a texture that must exist as long as the shape uses it. The shape doesn't store its own copy of the texture, but rather keeps a pointer to the texture. If the source texture is destroyed and the shape tries to use it, the behavior is undefined. The texture can be `NULL` to disable texturing. If `resetRect` is true, the `TextureRect` property of the shape is automatically adjusted to the size of the new texture. If it is false, the texture rect is left unchanged.

**Parameters**

- `texture`: New texture
- `resetRect`: Should the texture rect be reset to the size of the new texture?

**See also**

- `getTexture`, `setTextureRect`
Recompute the internal geometry of the shape.

This function must be called by the derived class every time the shape's points change (i.e., the result of either `getPointCount` or `getPoint` is different).

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

- `Shape.hpp`
sf::Sprite Class Reference

Drawable representation of a texture, with its own transformations, color, etc.

#include <Sprite.hpp>

Inheritance diagram for sf::Sprite:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sprite ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Sprite (const Texture &amp;texture)</strong></td>
<td>Construct the sprite from a source texture. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Sprite (const Texture &amp;texture, const IntRect &amp;rectangle)</strong></td>
<td>Construct the sprite from a sub-rectangle of a source texture.</td>
</tr>
<tr>
<td><strong>void setTexture (const Texture &amp;texture, bool resetRect=false)</strong></td>
<td>Change the source texture of the sprite. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void setTextureRect (const IntRect &amp;rectangle)</strong></td>
<td>Set the sub-rectangle of the texture that the sprite will display.</td>
</tr>
<tr>
<td><strong>void setColor (const Color &amp;color)</strong></td>
<td>Set the global color of the sprite. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const Texture * getTexture () const</strong></td>
<td>Get the source texture of the sprite. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const IntRect &amp; getTextureRect () const</strong></td>
<td>Get the sub-rectangle of the texture displayed by the sprite.</td>
</tr>
<tr>
<td><strong>const Color &amp; getColor () const</strong></td>
<td>Get the global color of the sprite. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>FloatRect getLocalBounds () const</strong></td>
<td>Get the local bounding rectangle of the entity. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>FloatRect getGlobalBounds () const</strong></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td><code>void setPosition (float x, float y)</code></td>
<td>set the position of the object More...</td>
</tr>
<tr>
<td><code>void setPosition (const Vector2f &amp;position)</code></td>
<td>set the position of the object More...</td>
</tr>
<tr>
<td><code>void setRotation (float angle)</code></td>
<td>set the orientation of the object More...</td>
</tr>
<tr>
<td><code>void setScale (float factorX, float factorY)</code></td>
<td>set the scale factors of the object More...</td>
</tr>
<tr>
<td><code>void setScale (const Vector2f &amp;factors)</code></td>
<td>set the scale factors of the object More...</td>
</tr>
<tr>
<td><code>void setOrigin (float x, float y)</code></td>
<td>set the local origin of the object More...</td>
</tr>
<tr>
<td><code>void setOrigin (const Vector2f &amp;origin)</code></td>
<td>set the local origin of the object More...</td>
</tr>
<tr>
<td><code>const Vector2f &amp; getPosition () const</code></td>
<td>get the position of the object More...</td>
</tr>
<tr>
<td><code>const Vector2f &amp; getRotation () const</code></td>
<td>get the orientation of the object More...</td>
</tr>
<tr>
<td><code>const Vector2f &amp; getScale () const</code></td>
<td>get the current scale of the object More...</td>
</tr>
<tr>
<td><code>const Vector2f &amp; getOrigin () const</code></td>
<td>get the local origin of the object More...</td>
</tr>
</tbody>
</table>
void move (float offsetX, float offsetY)
Move the object by a given offset. More...

void move (const Vector2f &offset)
Move the object by a given offset. More...

void rotate (float angle)
Rotate the object. More...

void scale (float factorX, float factorY)
Scale the object. More...

void scale (const Vector2f &factor)
Scale the object. More...

const Transform & getTransform () const
get the combined transform of the object More...

const Transform & getInverseTransform () const
get the inverse of the combined transform of the object More...
Detailed Description

**Drawable** representation of a texture, with its own transformations, color, etc.

`sf::Sprite` is a drawable class that allows to easily display a texture (or a part of it) on a render target.

It inherits all the functions from `sf::Transformable`: position, rotation, scale, properties such as the texture to use, the part of it to display, and some of the overall color of the sprite, or to get its bounding rectangle.

`sf::Sprite` works in combination with the `sf::Texture` class, which loads a given texture.

The separation of `sf::Sprite` and `sf::Texture` allows more flexibility and `sf::Texture` is a heavy resource, and any operation on it is slow (often too slow for real-time applications). On the other side, a `sf::Sprite` is a lightweight object which can use the pixel data with its own transformation/color/blending attributes.

It is important to note that the `sf::Sprite` instance doesn't copy the texture reference to it. Thus, a `sf::Texture` must not be destroyed while it is used by functions that uses a local `sf::Texture` instance for creating a sprite.

See also the note on coordinates and undistorted rendering in `sf::Transformable`.

Usage example:

```cpp
// Declare and load a texture
sf::Texture texture;
texture.loadFromFile("texture.png");

// Create a sprite
sf::Sprite sprite;
sprite.setTexture(texture);
sprite.setTextureRect(sf::IntRect(10, 10, 50, 30));
sprite.setColor(sf::Color(255, 255, 255, 200));
```
sprite.setPosition(100, 25);

// Draw it
window.draw(sprite);

See also
	sf::Texture, sf::Transformable

Definition at line 47 of file Sprite.hpp.
## Constructor & Destructor Documentation

### sf::Sprite::Sprite()

Default constructor.

Creates an empty sprite with no source texture.

### sf::Sprite::Sprite(const Texture & texture)

Construct the sprite from a source texture.

**Parameters**

- **texture** Source texture

**See also**

- setTexture

### sf::Sprite::Sprite(const Texture & texture, const IntRect & rectangle)

Construct the sprite from a sub-rectangle of a source texture.

**Parameters**

- **texture** Source texture
**rectangle** Sub-rectangle of the texture to assign to the sprite

See also

```
setTexture, setTextureRect
```
const Color& sf::Sprite::getColor ( ) const

Get the global color of the sprite.

Returns
  Global color of the sprite

See also
  setColor

FloatRect sf::Sprite::getGlobalBounds ( ) const

Get the global bounding rectangle of the entity.

The returned rectangle is in global coordinates, which means that it takes into account the transformations (translation, rotation, scale, ...) that are applied to the entity. In other words, this function returns the bounds of the sprite in the global 2D world's coordinate system.

Returns
  Global bounding rectangle of the entity

cost Transform& sf::Transformable::getInverseTransform ( ) const

get the inverse of the combined transform of the object
**Returns**

Inverse of the combined transformations applied to the object

**See also**

getTransform

---

**FloatRect** **sf::Sprite::getLocalBounds ( ) const**

Get the local bounding rectangle of the entity.

The returned rectangle is in local coordinates, which means that (translation, rotation, scale, ...) that are applied to the entity. In other bounds of the entity in the entity's coordinate system.

**Returns**

Local bounding rectangle of the entity

---

**const Vector2f& sf::Transformable::getOrigin ( ) const**

get the local origin of the object

**Returns**

Current origin

**See also**

setOrigin

---

**const Vector2f& sf::Transformable::getPosition ( ) const**
get the position of the object

**Returns**

Current position

**See also**

setPosition

---

```cpp
float sf::Transformable::getRotation() const
```

get the orientation of the object

The rotation is always in the range [0, 360].

**Returns**

Current rotation, in degrees

**See also**

setRotation

---

```cpp
const Vector2f& sf::Transformable::getScale() const
```

get the current scale of the object

**Returns**

Current scale factors

**See also**

setScale
**const Texture* sf::Sprite::getTexture ( ) const**

Get the source texture of the sprite.

If the sprite has no source texture, a NULL pointer is returned. This means that you can't modify the texture when you retrieve it with this function.

**Returns**
- Pointer to the sprite's texture

**See also**
- `setTexture`

**const IntRect& sf::Sprite::getTextureRect ( ) const**

Get the sub-rectangle of the texture displayed by the sprite.

**Returns**
- Texture rectangle of the sprite

**See also**
- `setTextureRect`

**const Transform& sf::Transformable::getTransform ( ) const**

get the combined transform of the object

**Returns**
- Transform combining the position/rotation/scale/origin of the object
void sf::Transformable::move(float offsetX, float offsetY)

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f pos = object.getPosition();
object.setPosition(pos.x + offsetX, pos.y + offsetY);
```

Parameters
- **offsetX** X offset
- **offsetY** Y offset

See also
- setPosition

---

void sf::Transformable::move(const Vector2f & offset)

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition equivalent to the following code:

```cpp
object.setPosition(object.getPosition() + offset);
```
Parameters
  offset Offset

See also
  setPosition

```cpp
template<typename T> object.setPosition(T pos);
```

### void sf::Transformable::rotate (float angle)

Rotate the object.

This function adds to the current rotation of the object, unlike `setRotation` which overwrites it. Thus, it is equivalent to the following code:

```
object.setRotation(object.getRotation() + angle);
```

Parameters
  angle Angle of rotation, in degrees

### void sf::Transformable::scale (float factorX, float factorY)

Scale the object.

This function multiplies the current scale of the object, unlike `setScale` which overwrites it. Thus, it is equivalent to the following code:

```
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factorX, scale.y * factorY);
```
**factorX** Horizontal scale factor

**factorY** Vertical scale factor

See also

setScale

---

```cpp
void sf::Transformable::scale (const Vector2f & factor)
```

Scale the object.

This function multiplies the current scale of the object, unlike `setScale` which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factor.x, scale.y * factor.y);
```

**Parameters**

- **factor** Scale factors

See also

setScale

---

```cpp
void sf::Sprite::setColor (const Color & color)
```

Set the global color of the sprite.

This color is modulated (multiplied) with the sprite's texture. It can be change its global opacity. By default, the sprite's color is opaque white.

**Parameters**

- **color** New color of the sprite
void sf::Transformable::setOrigin(float x, float y)

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

Parameters
- x X coordinate of the new origin
- y Y coordinate of the new origin

See also
- getColor

void sf::Transformable::setOrigin(const Vector2f & origin)

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

Parameters
- origin New origin

See also
- getOrigin
void sf::Transformable::setPosition ( float x, float y )

set the position of the object

This function completely overwrites the previous position. See the move function to apply an offset based on the previous position instead. The default position of a transformable object is (0, 0).

Parameters
- x X coordinate of the new position
- y Y coordinate of the new position

See also
- move, getPosition

void sf::Transformable::setPosition ( const Vector2f & position )

set the position of the object

This function completely overwrites the previous position. See the move function to apply an offset based on the previous position instead. The default position of a transformable object is (0, 0).

Parameters
- position New position

See also
- move, getPosition
void sf::Transformable::setRotation ( float angle )

set the orientation of the object

This function completely overwrites the previous rotation. See the rotate function on the previous rotation instead. The default rotation of a transformable object is 0.

Parameters
  
  **angle** New rotation, in degrees

See also
  
  rotate, getRotation

void sf::Transformable::setScale ( float factorX, float factorY )

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function on the previous scale instead. The default scale of a transformable object is (1, 1).

Parameters
  
  **factorX** New horizontal scale factor
  
  **factorY** New vertical scale factor

See also
  
  scale, getScale
void sf::Transformable::setScale ( const Vector2f &  factors )

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function to add a factor based on the previous scale instead. The default scale of a transformable object is (1, 1).

Parameters
   factors New scale factors

See also
   scale, getScale

void sf::Sprite::setTexture ( const Texture & texture,
                          bool                     resetRect = false )

Change the source texture of the sprite.

The texture argument refers to a texture that must exist as long as the sprite doesn't store its own copy of the texture, but rather keeps a pointer to it. If the source texture is destroyed and the sprite tries to use it, the TextureRect property of the sprite is automatically adjusted to the size of the new texture. If it is false, the texture rect is left unchanged.

Parameters
   texture New texture
   resetRect Should the texture rect be reset to the size of the new texture?

See also
   getTexture, setTextureRect
void sf::Sprite::setTextureRect ( const IntRect & rectangle )

Set the sub-rectangle of the texture that the sprite will display.

The texture rect is useful when you don't want to display the whole texture; by default, the texture rect covers the entire texture.

**Parameters**

- **rectangle** Rectangle defining the region of the texture to display

**See also**

- getTextureRect, setTexture

The documentation for this class was generated from the following file:
- Sprite.hpp
sf::Text Class Reference
Graphics module

Graphical text that can be drawn to a render target. More...

#include <Text.hpp>

Inheritance diagram for sf::Text:
## Public Types

<table>
<thead>
<tr>
<th>enum</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Regular = 0, <strong>Bold</strong> = 1 &lt;&lt; 0, <strong>Italic</strong> = 1 &lt;&lt; 1, <strong>Underlined</strong> = 1 &lt;&lt; 2, <strong>StrikeThrough</strong> = 1 &lt;&lt; 3</td>
</tr>
</tbody>
</table>

Enumeration of the string drawing styles. More...
Public Member Functions

<table>
<thead>
<tr>
<th>Function Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Text ()</code></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td><code>Text (const String &amp;string, const Font &amp;font, unsigned int characterSize=30)</code></td>
<td>Construct the text from a string, font and size. More...</td>
</tr>
<tr>
<td><code>void setString (const String &amp;string)</code></td>
<td>Set the text's string. More...</td>
</tr>
<tr>
<td><code>void setFont (const Font &amp;font)</code></td>
<td>Set the text's font. More...</td>
</tr>
<tr>
<td><code>void setCharacterSize (unsigned int size)</code></td>
<td>Set the character size. More...</td>
</tr>
<tr>
<td><code>void setStyle (Uint32 style)</code></td>
<td>Set the text's style. More...</td>
</tr>
<tr>
<td><code>void setColor (const Color &amp;color)</code></td>
<td>Set the fill color of the text. More...</td>
</tr>
<tr>
<td><code>void setFillColor (const Color &amp;color)</code></td>
<td>Set the fill color of the text. More...</td>
</tr>
<tr>
<td><code>void setOutlineColor (const Color &amp;color)</code></td>
<td>Set the outline color of the text. More...</td>
</tr>
<tr>
<td><code>void setOutlineThickness (float thickness)</code></td>
<td>Set the thickness of the text's outline. More...</td>
</tr>
<tr>
<td><code>const String &amp;(getString () const)</code></td>
<td>More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>const Font * getFont () const</code></td>
<td>Get the text's font.</td>
</tr>
<tr>
<td><code>unsigned int getCharacterSize () const</code></td>
<td>Get the character size.</td>
</tr>
<tr>
<td><code>Uint32 getStyle () const</code></td>
<td>Get the text's style.</td>
</tr>
<tr>
<td><code>const Color &amp; getColor () const</code></td>
<td>Get the fill color of the text.</td>
</tr>
<tr>
<td><code>const Color &amp; getFillColor () const</code></td>
<td>Get the fill color of the text.</td>
</tr>
<tr>
<td><code>const Color &amp; getOutlineColor () const</code></td>
<td>Get the outline color of the text.</td>
</tr>
<tr>
<td><code>float getOutlineThickness () const</code></td>
<td>Get the outline thickness of the text.</td>
</tr>
<tr>
<td><code>Vector2f findCharacterPos (std::size_t index) const</code></td>
<td>Return the position of the index-th character.</td>
</tr>
<tr>
<td><code>FloatRect getLocalBounds () const</code></td>
<td>Get the local bounding rectangle of the entity.</td>
</tr>
<tr>
<td><code>FloatRect getGlobalBounds () const</code></td>
<td>Get the global bounding rectangle of the entity.</td>
</tr>
<tr>
<td><code>void setPosition (float x, float y)</code></td>
<td>Set the position of the object.</td>
</tr>
</tbody>
</table>
void setPosition (const Vector2f &position)
set the position of the object More...

void setRotation (float angle)
set the orientation of the object More...

void setScale (float factorX, float factorY)
set the scale factors of the object More...

void setScale (const Vector2f &factors)
set the scale factors of the object More...

void setOrigin (float x, float y)
set the local origin of the object More...

void setOrigin (const Vector2f &origin)
set the local origin of the object More...

const Vector2f & getPosition () const
get the position of the object More...

float getRotation () const
get the orientation of the object More...

const Vector2f & getScale () const
get the current scale of the object More...

const Vector2f & getOrigin () const
get the local origin of the object More...

void move (float offsetX, float offsetY)
Move the object by a given offset. More...

void move (const Vector2f &offset)
Move the object by a given offset. More...
### void rotate (float angle)
- Rotate the object.  
  [More...](#)

### void scale (float factorX, float factorY)
- Scale the object.  
  [More...](#)

### void scale (const Vector2f &factor)
- Scale the object.  
  [More...](#)

### const Transform & getTransform () const
- get the combined transform of the object  
  [More...](#)

### const Transform & getInverseTransform () const
- get the inverse of the combined transform of the object
Detailed Description

Graphical text that can be drawn to a render target.

sf::Text is a drawable class that allows to easily display some text with custom style and color on a render target.

It inherits all the functions from sf::Transformable: position, rotation, scale properties such as the font to use, the character size, the font style (bold, italic, underlined, strike through), the global color and the text to display of course. It also provides convenience functions to calculate the graphical size of the text, or to get the global position of a given character.

sf::Text works in combination with the sf::Font class, which loads the glyphs (visual characters) of a given font.

The separation of sf::Font and sf::Text allows more flexibility and better performance: indeed, sf::Font is a heavy resource, and any operation on it is slow (often too slow for real-time applications). On the other side, a sf::Text is a lightweight object which can combine the glyphs of a font and display any text on a render target.

It is important to note that the sf::Text instance doesn't copy the font that it uses. Thus, a sf::Font must not be destroyed while it is used by a sf::Text (which uses a local sf::Font instance for creating a text).

See also the note on coordinates and undistorted rendering in sf::Transformable.

Usage example:

```cpp
// Declare and load a font
sf::Font font;
font.loadFromFile("arial.ttf");

// Create a text
```
sf::Text text("hello", font);
text.setCharacterSize(30);
text.setStyle(sf::Text::Bold);
text.setColor(sf::Color::Red);

// Draw it
window.draw(text);

See also
sf::Font, sf::Transformable

Definition at line 48 of file Text.hpp.
**Member Enumeration Documentation**

```markdown
**enum sf::Text::Style**

Enumeration of the string drawing styles.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>Regular characters, no style.</td>
</tr>
<tr>
<td>Bold</td>
<td>Bold characters.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic characters.</td>
</tr>
<tr>
<td>Underlined</td>
<td>Underlined characters.</td>
</tr>
<tr>
<td>StrikeThrough</td>
<td>Strike through characters.</td>
</tr>
</tbody>
</table>

Definition at line 56 of file `Text.hpp`.
```
### Constructor & Destructor Documentation

#### sf::Text::Text ( )

Default constructor.

Creates an empty text.

#### sf::Text::Text ( const String & string, const Font & font, unsigned int characterSize = 30 )

Construct the text from a string, font and size.

Note that if the used font is a bitmap font, it is not scalable, thus not all to use. This needs to be taken into consideration when setting the character text of a certain size, make sure the corresponding bitmap font that supp

**Parameters**

- **string**: Text assigned to the string
- **font**: Font used to draw the string
- **characterSize**: Base size of characters, in pixels
Member Function Documentation

**Vector2f sf::Text::findCharacterPos ( std::size_t index ) const**

Return the position of the *index*-th character.

This function computes the visual position of a character from its index; position is in global coordinates (translation, rotation, scale and origin range, the position of the end of the string is returned.

**Parameters**

- **index** Index of the character

**Returns**

Position of the character

**unsigned int sf::Text::getCharacterSize ( ) const**

Get the character size.

**Returns**

Size of the characters, in pixels

**See also**

- setCharacterSize

**const Color& sf::Text::getColor ( ) const**
Get the fill color of the text.

**Returns**

Fill color of the text

**See also**

setFillColor

**Deprecated:**

There is now fill and outline colors instead of a single global color. Use getOutlineColor() instead.

---

**const Color& sf::Text::getFillColor ( ) const**

Get the fill color of the text.

**Returns**

Fill color of the text

**See also**

setFillColor

---

**const Font* sf::Text::getFont ( ) const**

Get the text's font.

If the text has no font attached, a NULL pointer is returned. The return value that you cannot modify the font when you get it from this function.

**Returns**
Pointer to the text's font

**See also**
setFont

---

**FloatRect sf::Text::getGlobalBounds ( ) const**

Get the global bounding rectangle of the entity.

The returned rectangle is in global coordinates, which means it takes into account the transformations (translation, rotation, scale, ...) that are applied to the entity. It returns the bounds of the text in the global 2D world's coordinate system.

**Returns**
Global bounding rectangle of the entity

---

**const Transform& sf::Transformable::getInverseTransform ( ) const**

get the inverse of the combined transform of the object

**Returns**
Inverse of the combined transformations applied to the object

**See also**
getTransform

---

**FloatRect sf::Text::getLocalBounds ( ) const**

Get the local bounding rectangle of the entity.
The returned rectangle is in local coordinates, which means that (translation, rotation, scale, ...) that are applied to the entity. In other bounds of the entity in the entity's coordinate system.

**Returns**
Local bounding rectangle of the entity

```cpp
const Vector2f& sf::Transformable::getOrigin () const
```

get the local origin of the object

**Returns**
Current origin

**See also**
`setOrigin`

```cpp
const Color& sf::Text::getOutlineColor () const
```

Get the outline color of the text.

**Returns**
Outline color of the text

**See also**
`setOutlineColor`

```cpp
float sf::Text::getOutlineThickness () const
```
Get the outline thickness of the text.

**Returns**
Outline thickness of the text, in pixels

**See also**
setOutlineThickness

---

`const Vector2f& sf::Transformable::getPosition ( ) const`

get the position of the object

**Returns**
Current position

**See also**
setPosition

---

`float sf::Transformable::getRotation ( ) const`

get the orientation of the object

The rotation is always in the range [0, 360].

**Returns**
Current rotation, in degrees

**See also**
setRotation
**const Vector2f** & sf::Transformable::getScale ( ) const

get the current scale of the object

**Returns**

Current scale factors

**See also**

setScale

**const String** & sf::Text::getString ( ) const

Get the text's string.

The returned string is a sf::String, which can automatically be converted following lines of code are all valid:

```cpp
sf::String s1 = text.getString();
std::string s2 = text.getString();
std::wstring s3 = text.getString();
```

**Returns**

Text's string

**See also**

setString

**Uint32** sf::Text::getStyle ( ) const

Get the text's style.
Returns
   Text's style

See also
   setStyle

const Transform& sf::Transformable::getTransform ( ) const

get the combined transform of the object

Returns
   Transform combining the position/rotation/scale/origin of the object

See also
   getInverseTransform

void sf::Transformable::move ( float offsetX, float offsetY )

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition equivalent to the following code:

```
sf::Vector2f pos = object.getPosition();
object.setPosition(pos.x + offsetX, pos.y + offsetY);
```

Parameters
   offsetX X offset
   offsetY Y offset
void sf::Transformable::move ( const Vector2f & offset )

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setPosition(object.getPosition() + offset);
```

Parameters
---

offset Offset

See also
---

setPosition

---

void sf::Transformable::rotate ( float angle )

Rotate the object.

This function adds to the current rotation of the object, unlike setRotation which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setRotation(object.getRotation() + angle);
```

Parameters
---

angle Angle of rotation, in degrees

See also
---

setPosition
void sf::Transformable::scale ( float factorX, float factorY )

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factorX, scale.y * factorY);
```

Parameters

- **factorX** Horizontal scale factor
- **factorY** Vertical scale factor

See also

- setScale

---

void sf::Transformable::scale ( const Vector2f & factor )

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factor.x, scale.y * factor.y);
```

Parameters

- **factor** Scale factors
See also

setScale

**void sf::Text::setCharacterSize (unsigned int size)**

Set the character size.

The default size is 30.

Note that if the used font is a bitmap font, it is not scalable, thus not all to use. This needs to be taken into consideration when setting the character size of a certain size, make sure the corresponding bitmap font that supports that size is used.

**Parameters**

- **size** New character size, in pixels

See also

getCharacterSize

**void sf::Text::setColor (const Color & color)**

Set the fill color of the text.

By default, the text's fill color is opaque white. Setting the fill color to a transparent color will cause the outline to be displayed in the fill area of the text.

**Parameters**

- **color** New fill color of the text

See also

getFillColor
Deprecated:
There is now fill and outline colors instead of a single global color. Use `setOutlineColor()` instead.

### void sf::Text::setFillColor (const Color & color)

Set the fill color of the text.

By default, the text's fill color is opaque white. Setting the fill color to a transparent color will cause the outline to be displayed in the fill area of the text.

**Parameters**
- `color` New fill color of the text

**See also**
- `getFillColor`

### void sf::Text::setFont (const Font & font)

Set the text's font.

The `font` argument refers to a font that must exist as long as the text store its own copy of the font, but rather keeps a pointer to the one that font is destroyed and the text tries to use it, the behavior is undefined.

**Parameters**
- `font` New font

**See also**
- `getFont`
void sf::Transformable::setOrigin ( float x, float y )

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

Parameters

- x X coordinate of the new origin
- y Y coordinate of the new origin

See also
getOrigin

void sf::Transformable::setOrigin ( const Vector2f & origin )

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner and ignore all transformations (position, scale, rotation). The default origin of a transformable object is (0, 0).

Parameters

- origin New origin

See also
getOrigin
**void sf::Text::setOutlineColor ( const Color & color )**

Set the outline color of the text.

By default, the text's outline color is opaque black.

**Parameters**

- **color** New outline color of the text

**See also**

- getOutlineColor

---

**void sf::Text::setOutlineThickness ( float thickness )**

Set the thickness of the text's outline.

By default, the outline thickness is 0.

Be aware that using a negative value for the outline thickness will cause strange rendering.

**Parameters**

- **thickness** New outline thickness, in pixels

**See also**

- getOutlineThickness

---

**void sf::Transformable::setPosition ( float x, float y )**
set the position of the object

This function completely overwrites the previous position. See the move function on the previous position instead. The default position of a transformable object is (0,0).

**Parameters**

- **x** X coordinate of the new position
- **y** Y coordinate of the new position

**See also**

- move, getPosition

---

```cpp
void sf::Transformable::setPosition ( const Vector2f & position )
```

set the position of the object

This function completely overwrites the previous position. See the move function on the previous position instead. The default position of a transformable object is (0,0).

**Parameters**

- **position** New position

**See also**

- move, getPosition

---

```cpp
void sf::Transformable::setRotation ( float angle )
```

set the orientation of the object

This function completely overwrites the previous rotation. See the rotate function.
on the previous rotation instead. The default rotation of a transformable object is 0.

**Parameters**

- **angle** New rotation, in degrees

**See also**

- rotate, getRotation

```cpp
void sf::Transformable::setScale ( float factorX, float factorY )
```

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function to add a factor based on the previous scale instead. The default scale of a transformable object is (1, 1).

**Parameters**

- **factorX** New horizontal scale factor
- **factorY** New vertical scale factor

**See also**

- scale, getScale

```cpp
void sf::Transformable::setScale ( const Vector2f & factors )
```

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function to add a factor based on the previous scale instead. The default scale of a transformable object is (1, 1).
void sf::Text::setText(const String & string)

Set the text's string.

The string argument is a sf::String, which can automatically be constructed.
So, the following calls are all valid:

```cpp
sf::Text text;

text.setText("hello");
text.setText(L"hello");
text.setText(std::string("hello"));
text.setText(std::wstring(L"hello"));
```

A text's string is empty by default.

Parameters

- string New string

See also

getSetText

void sf::Text::setStyle(Uint32 style)

Set the text's style.

You can pass a combination of one or more styles, for example sf::Text::Bold style is sf::Text::Regular.
Parameters
   style New style

See also
   getStyle

The documentation for this class was generated from the following file:
   Text.hpp
sf::Texture Class Reference
Graphics module

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

#include <Texture.hpp>

Inheritance diagram for sf::Texture:
## Public Types

```c
enum CoordinateType { Normalized, Pixels }
```

Types of texture coordinates that can be used for rendering. More...
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Texture ()</strong></td>
<td>Default constructor.</td>
</tr>
<tr>
<td><strong>Texture (const Texture &amp;copy)</strong></td>
<td>Copy constructor.</td>
</tr>
<tr>
<td><strong>~Texture ()</strong></td>
<td>Destructor.</td>
</tr>
<tr>
<td><strong>bool create (unsigned int width, unsigned int height)</strong>**</td>
<td>Create the texture.</td>
</tr>
<tr>
<td><strong>bool loadFromFile (const std::string &amp;filename, const IntRect &amp;area)</strong></td>
<td>Load the texture from a file on disk.</td>
</tr>
<tr>
<td>*<em>bool loadFromMemory (const void <em>data, std::size_t size, const IntRect &amp;area)</em></em></td>
<td>Load the texture from a file in memory.</td>
</tr>
<tr>
<td><strong>bool loadFromStream (InputStream &amp;stream, const IntRect &amp;area)</strong></td>
<td>Load the texture from a custom stream.</td>
</tr>
<tr>
<td><strong>bool loadFromImage (const Image &amp;image, const IntRect &amp;area)</strong></td>
<td>Load the texture from an image.</td>
</tr>
<tr>
<td><strong>Vector2u getSize () const</strong></td>
<td>Return the size of the texture.</td>
</tr>
<tr>
<td><strong>Image copyToImage () const</strong></td>
<td>Copy the texture pixels to an image.</td>
</tr>
<tr>
<td>*<em>void update (const Uint8 <em>pixels)</em></em></td>
<td></td>
</tr>
</tbody>
</table>
### update

**Signature:**

```c
void update (const Uint8 *pixels, unsigned int width, unsigned int height, unsigned int x, unsigned int y)
```

Update the whole texture from an array of pixels. [More...](#)

---

### update

**Signature:**

```c
void update (const Image &image)
```

Update the texture from an image. [More...](#)

---

### update

**Signature:**

```c
void update (const Image &image, unsigned int x, unsigned int y)
```

Update a part of the texture from an image. [More...](#)

---

### update

**Signature:**

```c
void update (const Window &window)
```

Update the texture from the contents of a window. [More...](#)

---

### update

**Signature:**

```c
void update (const Window &window, unsigned int x, unsigned int y)
```

Update a part of the texture from the contents of a window. [More...](#)

---

### setSmooth

**Signature:**

```c
void setSmooth (bool smooth)
```

Enable or disable the smooth filter. [More...](#)

---

### isSmooth

**Signature:**

```c
bool isSmooth () const
```

Tell whether the smooth filter is enabled or not. [More...](#)

---

### setSrgb

**Signature:**

```c
void setSrgb (bool sRgb)
```

Enable or disable conversion from sRGB. [More...](#)

---

### isSrgb

**Signature:**

```c
bool isSrgb () const
```

Tell whether the texture source is converted from sRGB or not. [More...](#)

---

### setRepeateed

**Signature:**

```c
void setRepeateed (bool repeated)
```

Enable or disable repeating. [More...](#)

---

### isRepeateed

**Signature:**

```c
bool isRepeateed () const
```

Tell whether the texture is repeated or not. [More...](#)
bool generateMipmap ()
Generate a mipmap using the current texture data. More...

Texture & operator= (const Texture &right)
Overload of assignment operator. More...

unsigned int getNativeHandle () const
Get the underlying OpenGL handle of the texture. More...
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bind</code></td>
<td><code>static void bind (const Texture *texture, CoordinateType coordinateType)</code></td>
<td>Bind a texture for rendering. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>getMaximumSize</code></td>
<td><code>static unsigned int getMaximumSize ()</code></td>
<td>Get the maximum texture size allowed. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Static Private Member Functions

```c
static void ensureGLContext ()
Empty function for ABI compatibility, use acquireTransientContext
```

## Friends

<table>
<thead>
<tr>
<th>class</th>
<th>RenderTexture</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>RenderTarget</td>
</tr>
</tbody>
</table>
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 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 as a *sf::Image*, you need to prepare the pixels first and then upload operation (see `Texture::update`).

*sf::Texture* makes it easy to convert from/to *sf::Image*, but keep in mind between the graphics card and the central memory, therefore they are slow.

A texture can be loaded from an image, but also directly from a file. shortcuts are defined so that you don't need an image first for the most want to perform some modifications on the pixels before creating the final a *sf::Image*, do whatever you need with the pixels, and then call `Texture::loadFromImage()`.

Since they live in the graphics card memory, the pixels of a texture cannot first. And they cannot be accessed individually. Therefore, if you need to pixel-perfect collisions), it is recommended to store the collision information array of booleans.

Like *sf::Image*, *sf::Texture* can handle a unique internal representation of. This means that a pixel must be composed of 8 bits red, green, blue *sf::Color*.

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

// Load a texture from a file
sf::Texture texture;
if (!texture.loadFromFile("texture.png"))
    return -1;

// Assign it to a sprite
sf::Sprite sprite;
sprite.setTexture(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;
if (!texture.create(640, 480))
    return -1;

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

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

    // update the texture
    sf::Uint8* pixels = ...; // get a fresh chunk of pixels (the next frame)
    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(NULL);

See also
    sf::Sprite, sf::Image, sf::RenderTexture
Definition at line 47 of file Texture.hpp.
### Member Enumeration Documentation

**enum sf::Texture::CoordinateType**

Types of texture coordinates that can be used for rendering.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized</td>
<td>Texture coordinates in range [0 .. 1].</td>
</tr>
<tr>
<td>Pixels</td>
<td>Texture coordinates in range [0 .. size].</td>
</tr>
</tbody>
</table>

Definition at line 55 of file `Texture.hpp`. 
Constructor & Destructor Documentation

sf::Texture::Texture ( )

Default constructor.

Creates an empty texture.

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

Copy constructor.

Parameters

- **copy** instance to copy

sf::Texture::~Texture ( )

Destructor.
**Member Function Documentation**

```cpp
class sf::Texture
{
public:
  static void bind (const Texture * texture,
                    CoordinateType coordinateType = Normalized);
}
```

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.

```cpp
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(NULL);
// draw OpenGL stuff that use no texture...
```

The `coordinateType` argument controls how texture coordinates will be interpreted (the default is `Normalized`), 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

Copy the texture pixels to an image.

This function performs a slow operation that downloads the texture's pixels, copies them to a new image, potentially applying transformations to pixels (padded or flipped).

**Returns**
- **Image** containing the texture's pixels

**See also**
- loadFromImage

---

**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

**Returns**
- True if creation was successful

---

**bool** sf::Texture::generateMipmap ( )
Generate a mipmap using the current texture data.

Mipmaps are pre-computed chains of optimized textures. Each level of 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 filtering algorithms to improve the visual quality of textures when they are applied to objects much smaller than their original size. This process 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. Mipmaps are only valid from the time they are 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 ( )

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 cards, and up to 8192 pixels or more for newer hardware.

**Returns**

Maximum size allowed for textures, in pixels.

---

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

Get the underlying OpenGL handle of the texture.
You shouldn't need to use this function, unless you have very specific stuff 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

Return the size of the texture.

Returns
Size in pixels

bool sf::Texture::isRepeated () const

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

Tell whether the smooth filter is enabled or not.

Returns
True if smoothing is enabled, false if it is disabled
bool sf::Texture::isSrgb () const

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::string & filename,
const IntRect & area = IntRect())

Load the texture from a file on disk.

This function is a shortcut for the following code:

```cpp
sf::Image image;
image.loadFromFile(filename);
texture.loadFromImage(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 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

Returns

- True if loading was successful

See also

- `loadFromMemory`, `loadFromStream`, `loadFromImage`

```cpp
bool sf::Texture::loadFromImage ( const Image & image,
                                  const IntRect & area = IntRect ()
                                )
```

Load the texture from an image.

The **area** argument can be used to load only a sub-rectangle of the whole image. If the area is outside 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
- **area** Area of the image to load

Returns
True if loading was successful

See also
loadFromFile, loadFromMemory

```cpp
bool sf::Texture::loadFromMemory ( const void * data, std::size_t size, const IntRect & area = IntRect() )
```

Load the texture from a file in memory.

This function is a shortcut for the following code:

```cpp
sf::Image image;
image.loadFromMemory(data, size);
texture.loadFromImage(image, area);
```

The `area` argument can be used to load only a sub-rectangle of the whole image then leave the default value (which is an empty IntRect). If the `area` 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

**Returns**
bool sf::Texture::loadFromStream ( InputStream & stream,
    const IntRect & area = IntRect () )

Load the texture from a custom stream.

This function is a shortcut for the following code:

```
sf::Image image;
image.loadFromStream(stream);
texture.loadFromImage(image, area);
```

The `area` argument can be used to load only a sub-rectangle of the whole image then leave the default value (which is an empty IntRect). If the `area` 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

**Returns**

True if loading was successful

**See also**
Texture& sf::Texture::operator= ( const Texture & right )

Overload of assignment operator.

Parameters
right Instance to assign

Returns
Reference to self

void sf::Texture::setRepeated ( bool repeated )

Enable or disable repeating.

Repeating is involved when using texture coordinates outside the texture rectangle. In this case, if repeat mode is enabled, the whole texture will be repeated to reach the coordinate (for example, if the X texture coordinate is 3 * width 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. Repeating mode can be used reliably only if the texture has power-of-two dimensions (such as 256x128).

Parameters
repeated True to repeat the texture, false to disable repeating

See also
isRepeated

loadFromFile, loadFromMemory, loadFromImage
**void sf::Texture::setSmooth ( bool smooth )**

Enable or disable the smooth filter.

When the filter is activated, the texture appears smoother so that pixels you want the texture to look exactly the same as its source file, you should disable the filter by default.

**Parameters**

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

**See also**

- `isSmooth`

**void sf::Texture::setSrgb ( bool sRgb )**

Enable or disable conversion from sRGB.

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 to be "uncorrected" back to linear color space before being processed. 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.

After enabling or disabling sRGB conversion, make sure to reload the texture data in order for the setting to take effect.

This option is only useful in conjunction with an sRGB capable framebuffer window creation.

**Parameters**

- `sRgb` True to enable sRGB conversion, false to disable it
See also

isSrgb

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 pixels.

No additional check is performed on the size of the pixel array, passing undefined behavior.

This function does nothing if pixels is null or if the texture was not previou

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, ε pixels.

No additional check is performed on the size of the pixel array or the
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
- `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

```cpp
def sf::Texture::update(const sf::Image& image)
```

Update the texture from an image.

Although the source image can be smaller than the texture, this function
updates 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 behavior.

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

**Parameters**
- `image` Image to copy to the texture

```cpp
def sf::Texture::update(const sf::Image& 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 behavior.

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

**Parameters**
- `image` *Image* 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 & window )**

Update the texture from the contents of a window.

Although the source window can be smaller than the texture, this function updates 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 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, 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 a size and offset will lead to an undefined behavior.

This function does nothing if either the texture or the window was not pre

**Parameters**

- `window` Window 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`
Define a 3x3 transform matrix. More...

```
#include <Transform.hpp>
```
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform ()</td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td>Transform (float a00, float a01, float a02, float a10, float a11, float a12, float a20, float a21, float a22)</td>
<td>Construct a transform from a 3x3 matrix. More...</td>
</tr>
<tr>
<td>const float * getMatrix () const</td>
<td>Return the transform as a 4x4 matrix. More...</td>
</tr>
<tr>
<td>Transform getInverse () const</td>
<td>Return the inverse of the transform. More...</td>
</tr>
<tr>
<td>Vector2f transformPoint (float x, float y) const</td>
<td>Transform a 2D point. More...</td>
</tr>
<tr>
<td>Vector2f transformPoint (const Vector2f &amp;point) const</td>
<td>Transform a 2D point. More...</td>
</tr>
<tr>
<td>FloatRect transformRect (const FloatRect &amp;rectangle) const</td>
<td>Transform a rectangle. More...</td>
</tr>
<tr>
<td>Transform &amp; combine (const Transform &amp;transform)</td>
<td>Combine the current transform with another one. More...</td>
</tr>
<tr>
<td>Transform &amp; translate (float x, float y)</td>
<td>Combine the current transform with a translation. More...</td>
</tr>
<tr>
<td>Transform &amp; translate (const Vector2f &amp;offset)</td>
<td>Combine the current transform with a translation. More...</td>
</tr>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>Transform &amp; rotate (float angle)</td>
<td>Combine the current transform with a rotation. More...</td>
</tr>
<tr>
<td>Transform &amp; rotate (float angle, float centerX, float centerY)</td>
<td>Combine the current transform with a rotation. More...</td>
</tr>
<tr>
<td>Transform &amp; rotate (float angle, const Vector2f &amp;center)</td>
<td>Combine the current transform with a rotation. More...</td>
</tr>
<tr>
<td>Transform &amp; scale (float scaleX, float scaleY)</td>
<td>Combine the current transform with a scaling. More...</td>
</tr>
<tr>
<td>Transform &amp; scale (float scaleX, float scaleY, float centerX, float centerY)</td>
<td>Combine the current transform with a scaling. More...</td>
</tr>
<tr>
<td>Transform &amp; scale (const Vector2f &amp;factors)</td>
<td>Combine the current transform with a scaling. More...</td>
</tr>
<tr>
<td>Transform &amp; scale (const Vector2f &amp;factors, const Vector2f &amp;center)</td>
<td>Combine the current transform with a scaling. More...</td>
</tr>
</tbody>
</table>
# Static Public Attributes

<table>
<thead>
<tr>
<th>static const Transform Identity</th>
<th>The identity transform (does nothing)</th>
</tr>
</thead>
</table>

More...
## Related Functions

(Note that these are not member functions.)

<table>
<thead>
<tr>
<th>Type</th>
<th>Operator*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform</td>
<td>(const Transform &amp;left, const Transform &amp;right)</td>
<td>Overload of binary operator * to combine two transforms.</td>
</tr>
<tr>
<td>Transform &amp;</td>
<td>(Transform &amp;left, const Transform &amp;right)</td>
<td>Overload of binary operator *= to combine two transforms.</td>
</tr>
<tr>
<td>Vector2f</td>
<td>(const Transform &amp;left, const Vector2f &amp;right)</td>
<td>Overload of binary operator * to transform a point. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Define a 3x3 transform matrix.

A sf::Transform specifies how to translate, rotate, scale, shear, project, whatever.

In mathematical terms, it defines how to transform a coordinate system into another.

For example, if you apply a rotation transform to a sprite, the result will be a rotated sprite. Anything that is transformed by this rotation transform will be rotated the same way.

Transforms are typically used for drawing. But they can also be used for any computation that requires transforming points between the local and global coordinate systems of an entity.

Example:

```cpp
// define a translation transform
sf::Transform translation;
translation.translate(20, 50);

// define a rotation transform
sf::Transform rotation;
rotation.rotate(45);

// combine them
sf::Transform transform = translation * rotation;

// use the result to transform stuff...
sf::Vector2f point = transform.transformPoint(10, 20);
sf::FloatRect rect = transform.transformRect(sf::FloatRect(0, 0, 10, 100));
```

See also

sf::Transformable, sf::RenderStates

Definition at line 42 of file Transform.hpp.
Constructor & Destructor Documentation

_sf::Transform::Transform ( )_

Default constructor.

Creates an identity transform (a transform that does nothing).

_sf::Transform::Transform ( float a00, float a01, float a02, float a10, float a11, float a12, float a20, float a21, float a22 )_

Construct a transform from a 3x3 matrix.

**Parameters**

- _a00_ Element (0, 0) of the matrix
- _a01_ Element (0, 1) of the matrix
- _a02_ Element (0, 2) of the matrix
- _a10_ Element (1, 0) of the matrix
- _a11_ Element (1, 1) of the matrix

\textbf{a_{12}} Element (1, 2) of the matrix
\textbf{a_{20}} Element (2, 0) of the matrix
\textbf{a_{21}} Element (2, 1) of the matrix
\textbf{a_{22}} Element (2, 2) of the matrix
Member Function Documentation

**Transform** & sf::Transform::combine (const Transform & transform)

Combine the current transform with another one.

The result is a transform that is equivalent to applying *this followed by equivalent to a matrix multiplication.

**Parameters**
- **transform** Transform to combine with this transform

**Returns**
- Reference to *this

**Transform** sf::Transform::getInverse () const

Return the inverse of the transform.

If the inverse cannot be computed, an identity transform is returned.

**Returns**
- A new transform which is the inverse of self

**const float** * sf::Transform::getMatrix () const

Return the transform as a 4x4 matrix.

This function returns a pointer to an array of 16 floats containing the transform, which is directly compatible with OpenGL functions.

```cpp
sf::Transform transform = ...;
glLoadMatrixf(transform.getMatrix());
```

**Returns**

Pointer to a 4x4 matrix

---

**Transform** & *sf::Transform::rotate (float angle)*

Combine the current transform with a rotation.

This function returns a reference to *this, so that calls can be chained.

```cpp
sf::Transform transform;
transform.rotate(90).translate(50, 20);
```

**Parameters**

- **angle** Rotation angle, in degrees

**Returns**

Reference to *this

**See also**

translate, scale

---

**Transform** & *sf::Transform::rotate (float angle, float centerX, float centerY)*

**Parameters**

- **angle** Rotation angle, in degrees
- **centerX** Translation along the x-axis
- **centerY** Translation along the y-axis

**Returns**

Reference to *this
float centerY
)

Combine the current transform with a rotation.

The center of rotation is provided for convenience as a second argument, so that you can build rotations around arbitrary points more easily (and efficiently) than translate(-center).rotate(angle).translate(center).

This function returns a reference to *this, so that calls can be chained.

```
sf::Transform transform;
transform.rotate(90, 8, 3).translate(50, 20);
```

**Parameters**

- **angle** Rotation angle, in degrees
- **centerX** X coordinate of the center of rotation
- **centerY** Y coordinate of the center of rotation

**Returns**

Reference to *this

**See also**

translate, scale

This function returns a reference to *this, so that calls can be chained.

```cpp
sf::Transform transform;
transform.rotate(90, sf::Vector2f(8, 3)).translate(sf::Vector2f(50, 2));
```

### Parameters
- **angle** Rotation angle, in degrees
- **center** Center of rotation

### Returns
Reference to *this

### See also
- `translate`, `scale`

---

```cpp
sf::Transform& sf::Transform::scale ( float scaleX, float scaleY )
```

Combine the current transform with a scaling.

This function returns a reference to *this, so that calls can be chained.

```cpp
sf::Transform transform;
transform.scale(2, 1).rotate(45);
```

### Parameters
- **scaleX** Scaling factor on the X axis
- **scaleY** Scaling factor on the Y axis

### Returns
Reference to *this

See also
translate, rotate

Transform& sf::Transform::scale ( float scaleX,  
    float scaleY,  
    float centerX,  
    float centerY  
)

Combine the current transform with a scaling.

The center of scaling is provided for convenience as a second argument around arbitrary points more easily (and efficiently) than translate(-center).scale(factors).translate(center).

This function returns a reference to *this, so that calls can be chained.

sf::Transform transform;  
transform.scale(2, 1, 8, 3).rotate(45);

Parameters

- **scaleX**  Scaling factor on X axis
- **scaleY**  Scaling factor on Y axis
- **centerX**  X coordinate of the center of scaling
- **centerY**  Y coordinate of the scaling

Returns

Reference to *this

See also
translate, rotate
**Transform** & `sf::Transform::scale (const Vector2f & factors)`

Combine the current transform with a scaling.

This function returns a reference to *this, so that calls can be chained.

```cpp
sf::Transform transform;
transform.scale(sf::Vector2f(2, 1)).rotate(45);
```

**Parameters**
- `factors` Scaling factors

**Returns**
- Reference to *this

**See also**
- `translate`, `rotate`

**Transform** & `sf::Transform::scale (const Vector2f & factors, const Vector2f & center)`

Combine the current transform with a scaling.

The center of scaling is provided for convenience as a second argument around arbitrary points more easily (and efficiently) than `translatex(center).scale(factors).translate(center)`.

This function returns a reference to *this, so that calls can be chained.

```cpp
sf::Transform transform;
```
transform.scale(sf::Vector2f(2, 1), sf::Vector2f(8, 3)).rotate(45);

**Parameters**
- **factors** Scaling factors
- **center** Center of scaling

**Returns**
- Reference to *this

**See also**
- translate, rotate

```cpp
sf::Transform::transformPoint (float x, float y) const
```

Transform a 2D point.

**Parameters**
- **x** X coordinate of the point to transform
- **y** Y coordinate of the point to transform

**Returns**
- Transformed point

```cpp
sf::Transform::transformPoint (const Vector2f & point) const
```

Transform a 2D point.

**Parameters**
- **point** Point to transform
Returns

Transformed point

FloatRect sf::Transform::transformRect ( const FloatRect & rectangle )

Transform a rectangle.

Since SFML doesn't provide support for oriented rectangles, the result is an axis-aligned rectangle. Which means that if the transform contains a rotation, the bounding rectangle of the transformed rectangle is returned.

Parameters

rectangle Rectangle to transform

Returns

Transformed rectangle

Transform& sf::Transform::translate ( float x, float y )

Combine the current transform with a translation.

This function returns a reference to *this, so that calls can be chained.

sf::Transform transform;
transform.translate(100, 200).rotate(45);

Parameters

x Offset to apply on X axis
y Offset to apply on Y axis
Transform\& sf::Transform::translate ( const Vector2f \& \_offset )

Combine the current transform with a translation.

This function returns a reference to *this, so that calls can be chained.

```cpp
sf::Transform transform;
transform.translate(sf::Vector2f(100, 200)).rotate(45);
```

**Parameters**

- **offset** Translation offset to apply

**Returns**

Reference to *this

**See also**

rotate, scale
**Transform operator** *( const Transform &  left,  
const Transform &  right  
)*

Overload of binary operator * to combine two transforms.

This call is equivalent to calling `Transform(left).combine(right)`.

**Parameters**

- **left**  Left operand (the first transform)
- **right**  Right operand (the second transform)

**Returns**

New combined transform

---

**Vector2f operator** *( const Transform &  left,  
const Vector2f &  right  
)*

Overload of binary operator * to transform a point.

This call is equivalent to calling `left.transformPoint(right)`.

**Parameters**

- **left**  Left operand (the transform)
- **right**  Right operand (the point to transform)
Returns

New transformed point

\textbf{Transform} \& \text{operator*= ( Transform \& left, const Transform \& right )}

Overload of binary operator *= to combine two transforms.

This call is equivalent to calling left.combine(right).

\textbf{Parameters}

- \textbf{left} Left operand (the first transform)
- \textbf{right} Right operand (the second transform)

\textbf{Returns}

The combined transform
**Member Data Documentation**

### `const Transform sf::Transform::Identity`

The identity transform (does nothing)

Definition at line 354 of file `Transform.hpp`.

The documentation for this class was generated from the following file:
- `Transform.hpp`
sf::Transformable Class Reference
Graphics module

Decomposed transform defined by a position, a rotation and a scale. More...

```
#include <Transformable.hpp>
```

Inheritance diagram for sf::Transformable:
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformable ()</td>
<td>Default constructor.</td>
<td>More...</td>
</tr>
<tr>
<td>virtual ~Transformable ()</td>
<td>Virtual destructor.</td>
<td>More...</td>
</tr>
<tr>
<td>void setPosition (float x, float y)</td>
<td>set the position of the object</td>
<td>More...</td>
</tr>
<tr>
<td>void setPosition (const Vector2f &amp;position)</td>
<td>set the position of the object</td>
<td>More...</td>
</tr>
<tr>
<td>void setRotation (float angle)</td>
<td>set the orientation of the object</td>
<td>More...</td>
</tr>
<tr>
<td>void setScale (float factorX, float factorY)</td>
<td>set the scale factors of the object</td>
<td>More...</td>
</tr>
<tr>
<td>void setScale (const Vector2f &amp;factors)</td>
<td>set the scale factors of the object</td>
<td>More...</td>
</tr>
<tr>
<td>void setOrigin (float x, float y)</td>
<td>set the local origin of the object</td>
<td>More...</td>
</tr>
<tr>
<td>void setOrigin (const Vector2f &amp;origin)</td>
<td>set the local origin of the object</td>
<td>More...</td>
</tr>
<tr>
<td>const Vector2f &amp; getPosition () const</td>
<td>get the position of the object</td>
<td>More...</td>
</tr>
<tr>
<td>float getRotation () const</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>getOrientation()</code></td>
<td>get the orientation of the object</td>
<td></td>
</tr>
<tr>
<td><code>getScale()</code></td>
<td>get the current scale of the object</td>
<td></td>
</tr>
<tr>
<td><code>getOrigin()</code></td>
<td>get the local origin of the object</td>
<td></td>
</tr>
<tr>
<td><code>move(float offsetX, float offsetY)</code></td>
<td>Move the object by a given offset.</td>
<td></td>
</tr>
<tr>
<td><code>move(const Vector2f &amp;offset)</code></td>
<td>Move the object by a given offset.</td>
<td></td>
</tr>
<tr>
<td><code>rotate(float angle)</code></td>
<td>Rotate the object.</td>
<td></td>
</tr>
<tr>
<td><code>scale(float factorX, float factorY)</code></td>
<td>Scale the object.</td>
<td></td>
</tr>
<tr>
<td><code>scale(const Vector2f &amp;factor)</code></td>
<td>Scale the object.</td>
<td></td>
</tr>
<tr>
<td><code>getTransform()</code></td>
<td>get the combined transform of the object</td>
<td></td>
</tr>
<tr>
<td><code>getInverseTransform()</code></td>
<td>get the inverse of the combined transform of the object</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Decomposed transform defined by a position, a rotation and a scale.

This class is provided for convenience, on top of sf::Transform.

sf::Transform, as a low-level class, offers a great level of flexibility but requires a bit of management. Indeed, one can easily combine any kind of operation, such as a translation followed by a scaling, but once the result transform is built, there's no way to go backward and change only the rotation without modifying the translation and scaling. Once the result transform is recomputed, which means that you need to retrieve the initial translation and scaling factors and combine them the same way you did before updating the rotation. This requires to store all the individual components of the final transform.

That's exactly what sf::Transformable was written for: it hides these variables behind an easy to use interface. You can set or get any of the individual components without worrying about the others. It also provides the composed transform (as a sf::Transform).

In addition to the position, rotation and scale, sf::Transformable provides an "origin" component, which represents the local origin of the three other components. Let's take an example: By default, the sprite is positioned/rotated/scaled relatively to its top-left corner (0, 0). But if we change the origin to be (5, 5), the sprite will be positioned/rotated/scaled around its center instead. And if we set the origin to (10, 10), it will be transformed around its bottom-right corner.

To keep the sf::Transformable class simple, there's only one origin for position the sprite relatively to its top-left corner while rotating it around it.

sf::Transformable can be used as a base class. It is often combined with sprites, texts and shapes do.
class MyEntity : public sf::Transformable, public sf::Drawable
{
  virtual void draw(sf::RenderTarget& target, sf::RenderStates states) {
    states.transform *= getTransform();
    target.draw(..., states);
  }
};

MyEntity entity;
entity.setPosition(10, 20);
entity.setRotation(45);
window.draw(entity);

It can also be used as a member, if you don't want to use its API directly functions, or you have different naming conventions for example).

class MyEntity
{
  public:
    void SetPosition(const MyVector& v)
    {
      myTransform.setPosition(v.x(), v.y());
    }

    void Draw(sf::RenderTarget& target) const
    {
      target.draw(..., myTransform.getTransform());
    }
  private:
    sf::Transformable myTransform;
};

A note on coordinates and undistorted rendering:
By default, SFML (or more exactly, OpenGL) may interpolate drawable when rendering. While this allows transitions like slow movements or rotations to appear smoothly, it can lead to unwanted results in some cases, for example blurred or distorted objects. To render a sf::Drawable object pixel-perfectly, make sure the involved coordinates allow a 1:1 mapping of pixels in the window to texels (pixels in the texture). More specifically, this means:

- The object's position, origin and scale have no fractional part
- The object's and the view's rotation are a multiple of 90 degrees
The view's center and size have no fractional part

See also

sf::Transform

Definition at line 41 of file Transformable.hpp.
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>sf::Transformable::Transformable ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default constructor.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual sf::Transformable::~Transformable ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual destructor.</td>
</tr>
</tbody>
</table>
### Member Function Documentation

**const Transform& sf::Transformable::getInverseTransform ( ) const**

get the inverse of the combined transform of the object

**Returns**

Inverse of the combined transformations applied to the object

**See also**

getTransform

**const Vector2f& sf::Transformable::getOrigin ( ) const**

get the local origin of the object

**Returns**

Current origin

**See also**

setOrigin

**const Vector2f& sf::Transformable::getPosition ( ) const**

get the position of the object
Returns
Current position

See also
setPosition

float sf::Transformable::getRotation ( ) const

get the orientation of the object

The rotation is always in the range [0, 360].

Returns
Current rotation, in degrees

See also
setRotation

const Vector2f& sf::Transformable::getScale ( ) const

get the current scale of the object

Returns
Current scale factors

See also
setScale

const Transform& sf::Transformable::getTransform ( ) const
get the combined transform of the object

**Returns**

*Transform* combining the position/rotation/scale/origin of the object

**See also**

getInverseTransform

```cpp
void sf::Transformable::move ( float offsetX, float offsetY )
```

Move the object by a given offset.

This function adds to the current position of the object, unlike setPosition which overwrites it. Thus, it is equivalent to the following code:

```cpp
sf::Vector2f pos = object.getPosition();
object.setPosition(pos.x + offsetX, pos.y + offsetY);
```

**Parameters**

- **offsetX** X offset
- **offsetY** Y offset

**See also**

setPosition

```cpp
void sf::Transformable::move ( const Vector2f & offset )
```

Move the object by a given offset.
This function adds to the current position of the object, unlike setPosition which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setPosition(object.getPosition() + offset);
```

**Parameters**
- **offset** Offset

**See also**
- setPosition

---

**void sf::Transformable::rotate (float angle)**

Rotate the object.

This function adds to the current rotation of the object, unlike setRotation which overwrites it. Thus, it is equivalent to the following code:

```cpp
object.setRotation(object.getRotation() + angle);
```

**Parameters**
- **angle** Angle of rotation, in degrees

---

**void sf::Transformable::scale (float factorX, float factorY)**

Scale the object.

This function multiplies the current scale of the object, unlike setScale which overwrites it. Thus, it is equivalent to the following code:
sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factorX, scale.y * factorY);

Parameters

- factorX Horizontal scale factor
- factorY Vertical scale factor

See also

setScale

---

void sf::Transformable::scale ( const Vector2f & factor )

Scale the object.

This function multiplies the current scale of the object, unlike setScale, equivalent to the following code:

sf::Vector2f scale = object.getScale();
object.setScale(scale.x * factor.x, scale.y * factor.y);

Parameters

- factor Scale factors

See also

setScale

---

void sf::Transformable::setOrigin ( float x, float y )
set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object. The default origin of a transformable object is (0, 0).

**Parameters**
- **x** X coordinate of the new origin
- **y** Y coordinate of the new origin

**See also**
- getOrigin

---

```cpp
void sf::Transformable::setOrigin ( const Vector2f & origin )
```

set the local origin of the object

The origin of an object defines the center point for all transformations. The coordinates of this point must be relative to the top-left corner of the object. The default origin of a transformable object is (0, 0).

**Parameters**
- **origin** New origin

**See also**
- getOrigin

---

```cpp
void sf::Transformable::setPosition ( float x, float y )
```
set the position of the object

This function completely overwrites the previous position. See the move function on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**
- `x` X coordinate of the new position
- `y` Y coordinate of the new position

**See also**
- `move`
- `getPosition`

```cpp
void sf::Transformable::setPosition ( const Vector2f & position )
```

set the position of the object

This function completely overwrites the previous position. See the move function on the previous position instead. The default position of a transformable object is (0, 0).

**Parameters**
- `position` New position

**See also**
- `move`
- `getPosition`

```cpp
void sf::Transformable::setRotation ( float angle )
```

set the orientation of the object

This function completely overwrites the previous rotation. See the rotate...
on the previous rotation instead. The default rotation of a transformable object is 0.

**Parameters**

- `angle` New rotation, in degrees

**See also**

- `rotate`, `getRotation`

```cpp
void sf::Transformable::setScale ( float factorX,
                                float factorY )
```

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function to add a factor based on the previous scale instead. The default scale of a transformable object is (1, 1).

**Parameters**

- `factorX` New horizontal scale factor
- `factorY` New vertical scale factor

**See also**

- `scale`, `getScale`

```cpp
void sf::Transformable::setScale ( const Vector2f & factors )
```

set the scale factors of the object

This function completely overwrites the previous scale. See the scale function to add a factor based on the previous scale instead. The default scale of a transformable object is (1, 1).
Parameters

**factors** New scale factors

See also

- scale, getScale

The documentation for this class was generated from the following file:
- Transformable.hpp
sf::Vertex Class Reference

Graphics module

Define a point with color and texture coordinates. More...

#include <Vertex.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Vertex()</code></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td><code>Vertex(const Vector2f &amp;thePosition)</code></td>
<td>Construct the vertex from its position. More...</td>
</tr>
<tr>
<td><code>Vertex(const Vector2f &amp;thePosition, const Color &amp;theColor)</code></td>
<td>Construct the vertex from its position and color. More...</td>
</tr>
<tr>
<td><code>Vertex(const Vector2f &amp;thePosition, const Vector2f &amp;theTexCoords)</code></td>
<td>Construct the vertex from its position and texture coordinates. More...</td>
</tr>
<tr>
<td><code>Vertex(const Vector2f &amp;thePosition, const Color &amp;theColor, const Vector2f &amp;theTexCoords)</code></td>
<td>Construct the vertex from its position, color and texture coordinates. More...</td>
</tr>
</tbody>
</table>
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector2f</td>
<td>position</td>
<td>2D position of the vertex</td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>color</td>
<td>Color of the vertex</td>
<td></td>
</tr>
<tr>
<td>Vector2f</td>
<td>texCoords</td>
<td>Coordinates of the texture's pixel to map to the vertex</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Define a point with color and texture coordinates.

A vertex is an improved point.

It has a position and other extra attributes that will be used for drawing: in and a pair of texture coordinates.

The vertex is the building block of drawing. Everything which is visible on are grouped as 2D primitives (triangles, quads, ...), and these primitives complex 2D entities such as sprites, texts, etc.

If you use the graphical entities of SFML (sprite, text, shape) you won't have to deal with vertices directly. But if you want to define your own 2D entities, such as tiled maps or particle systems, using vertices will allow you to get maximum performances.

Example:

```cpp
// define a 100x100 square, red, with a 10x10 texture mapped on it
sf::Vertex vertices[] =
{
    sf::Vertex(sf::Vector2f( 0, 0), sf::Color::Red, sf::Vector2f( 0, 0)),
    sf::Vertex(sf::Vector2f( 0, 100), sf::Color::Red, sf::Vector2f( 0, 10)),
    sf::Vertex(sf::Vector2f(100, 100), sf::Color::Red, sf::Vector2f(10, 1)),
    sf::Vertex(sf::Vector2f(100, 0), sf::Color::Red, sf::Vector2f(10, 0))
};
// draw it
window.draw(vertices, 4, sf::Quads);
```

Note: although texture coordinates are supposed to be an integer amount because of some buggy graphics drivers that are not able to process integer coordinates of some buggy graphics drivers that are not able to process integer coordinates, their type is float because of some buggy graphics drivers that are not able to process integer coordinates.

See also
sf::VertexArray

Definition at line 42 of file Vertex.hpp.
# Constructor & Destructor Documentation

## sf::Vertex::Vertex ( )

Default constructor.

## sf::Vertex::Vertex ( const Vector2f & thePosition )

Construct the vertex from its position.

The vertex color is white and texture coordinates are (0, 0).

### Parameters

- **thePosition** **Vertex** position

## sf::Vertex::Vertex ( const Vector2f & thePosition, const Color & theColor )

Construct the vertex from its position and color.

The texture coordinates are (0, 0).

### Parameters

- **thePosition** **Vertex** position
- **theColor** **Vertex** color
sf::Vertex::Vertex ( const Vector2f & thePosition,  
                 const Vector2f & theTexCoords )

Construct the vertex from its position and texture coordinates.

The vertex color is white.

Parameters
  thePosition  Vertex position
  theTexCoords Vertex texture coordinates

sf::Vertex::Vertex ( const Vector2f & thePosition,  
                 const Color & theColor,  
                 const Vector2f & theTexCoords )

Construct the vertex from its position, color and texture coordinates.

Parameters
  thePosition  Vertex position
  theColor     Vertex color
  theTexCoords Vertex texture coordinates
Member Data Documentation

<table>
<thead>
<tr>
<th><strong>Color</strong> sf::Vertex::color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color of the vertex.</td>
</tr>
<tr>
<td>Definition at line 98 of file Vertex.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Vector2f</strong> sf::Vertex::position</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D position of the vertex</td>
</tr>
<tr>
<td>Definition at line 97 of file Vertex.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Vector2f</strong> sf::Vertex::texCoords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinates of the texture's pixel to map to the vertex.</td>
</tr>
<tr>
<td>Definition at line 99 of file Vertex.hpp.</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following file:
- Vertex.hpp
Define a set of one or more 2D primitives. More...

#include <VertexArray.hpp>

Inheritance diagram for sf::VertexArray:
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VertexArray ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>VertexArray (PrimitiveType type, std::size_t vertexCount)</strong></td>
<td>Construct the vertex array with a type and an initial number of vertices.</td>
</tr>
<tr>
<td><strong>std::size_t getVertexCount () const</strong></td>
<td>Return the vertex count. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Vertex &amp; operator[] (std::size_t index)</strong></td>
<td>Get a read-write access to a vertex by its index. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>const Vertex &amp; operator[] (std::size_t index) const</strong></td>
<td>Get a read-only access to a vertex by its index. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void clear ()</strong></td>
<td>Clear the vertex array. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void resize (std::size_t vertexCount)</strong></td>
<td>Resize the vertex array. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void append (const Vertex &amp;vertex)</strong></td>
<td>Add a vertex to the array. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void setPrimitiveType (PrimitiveType type)</strong></td>
<td>Set the type of primitives to draw. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>PrimitiveType getPrimitiveType () const</strong></td>
<td>Get the type of primitives drawn by the vertex array. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>FloatRect getBounds () const</strong></td>
<td></td>
</tr>
</tbody>
</table>
Compute the bounding rectangle of the vertex array.
Detailed Description

Define a set of one or more 2D primitives.

sf::VertexArray is a very simple wrapper around a dynamic array of vertices.

It inherits sf::Drawable, but unlike other drawables it is not transformable.

Example:

```cpp
sf::VertexArray lines(sf::LineStrip, 4);
lines[0].position = sf::Vector2f(10, 0);
lines[1].position = sf::Vector2f(20, 0);
lines[2].position = sf::Vector2f(30, 5);
lines[3].position = sf::Vector2f(40, 2);
window.draw(lines);
```

See also

sf::Vertex

Definition at line 45 of file VertexArray.hpp.
## Constructor & Destructor Documentation

### sf::VertexArray::VertexArray ( )

**Default constructor.**

Creates an empty vertex array.

### sf::VertexArray::VertexArray ( PrimitiveType type, std::size_t vertexCount = 0 )

Construct the vertex array with a type and an initial number of vertices.

**Parameters**

- **type**  
  Type of primitives

- **vertexCount**  
  Initial number of vertices in the array
# Member Function Documentation

## `void sf::VertexArray::append ( const Vertex & vertex )`

Add a vertex to the array.

**Parameters**

- `vertex` `Vertex` to add

## `void sf::VertexArray::clear ( )`

Clear the vertex array.

This function removes all the vertices from the array. It doesn't deallocate that adding new vertices after clearing doesn't involve reallocating all the

## `FloatRect sf::VertexArray::getBounds ( ) const`

Compute the bounding rectangle of the vertex array.

This function returns the minimal axis-aligned rectangle that contains all the

**Returns**

- Bounding rectangle of the vertex array
**PrimitiveType** sf::VertexArray::getPrimitiveType ( ) const

Get the type of primitives drawn by the vertex array.

Returns

Primitive type

**std::size_t** sf::VertexArray::getVertexCount ( ) const

Return the vertex count.

Returns

Number of vertices in the array

**Vertex&** sf::VertexArray::operator[]( std::size_t index )

Get a read-write access to a vertex by its index.

This function doesn't check *index*, it must be in range [0, getVertexCount()] undefined otherwise.

Parameters

*index* Index of the vertex to get

Returns

Reference to the index-th vertex

See also

getVertexCount
**const Vertex& sf::VertexArray::operator[](std::size_t index) const**

Get a read-only access to a vertex by its index.

This function doesn't check `index`, it must be in range [0, `getVertexCount()`] undefined otherwise.

**Parameters**

- `index` Index of the vertex to get

**Returns**

Const reference to the index-th vertex

**See also**

- `getVertexCount`

---

**void sf::VertexArray::resize(std::size_t vertexCount)**

Resize the vertex array.

If `vertexCount` is greater than the current size, the previous vertices are kept and new (default-constructed) vertices are added. If `vertexCount` is less than the current size, existing vertices are removed from the array.

**Parameters**

- `vertexCount` New size of the array (number of vertices)

---

**void sf::VertexArray::setPrimitiveType(PrimitiveType type)**

Set the type of primitives to draw.
This function defines how the vertices must be interpreted when it's time

- As points
- As lines
- As triangles
- As quads The default primitive type is sf::Points.

**Parameters**

`type` Type of primitive

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

- VertexArray.hpp
sf::View Class Reference
Graphics module

2D camera that defines what region is shown on screen. More...

#include <View.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>View ()</strong></td>
<td>Default constructor.</td>
<td></td>
</tr>
<tr>
<td><strong>View (const FloatRect &amp;rectangle)</strong></td>
<td>Construct the view from a rectangle.</td>
<td></td>
</tr>
<tr>
<td><strong>View (const Vector2f &amp;center, const Vector2f &amp;size)</strong></td>
<td>Construct the view from its center and size.</td>
<td></td>
</tr>
<tr>
<td><strong>void setCenter (float x, float y)</strong></td>
<td>Set the center of the view.</td>
<td></td>
</tr>
<tr>
<td><strong>void setCenter (const Vector2f &amp;center)</strong></td>
<td>Set the center of the view.</td>
<td></td>
</tr>
<tr>
<td><strong>void setSize (float width, float height)</strong></td>
<td>Set the size of the view.</td>
<td></td>
</tr>
<tr>
<td><strong>void setSize (const Vector2f &amp;size)</strong></td>
<td>Set the size of the view.</td>
<td></td>
</tr>
<tr>
<td><strong>void setRotation (float angle)</strong></td>
<td>Set the orientation of the view.</td>
<td></td>
</tr>
<tr>
<td><strong>void setViewport (const FloatRect &amp;viewport)</strong></td>
<td>Set the target viewport.</td>
<td></td>
</tr>
<tr>
<td><strong>void reset (const FloatRect &amp;rectangle)</strong></td>
<td>Reset the view to the given rectangle.</td>
<td></td>
</tr>
<tr>
<td><strong>const Vector2f &amp; getCenter () const</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Get the center of the view. More...

**const Vector2f & getSize () const**
Get the size of the view. More...

**float getRotation () const**
Get the current orientation of the view. More...

**const FloatRect & getViewport () const**
Get the target viewport rectangle of the view. More...

**void move (float offsetX, float offsetY)**
Move the view relatively to its current position. More.

**void move (const Vector2f &offset)**
Move the view relatively to its current position. More.

**void rotate (float angle)**
Rotate the view relatively to its current orientation. More.

**void zoom (float factor)**
Resize the view rectangle relatively to its current size.

**const Transform & getTransform () const**
Get the projection transform of the view. More...

**const Transform & getInverseTransform () const**
Get the inverse projection transform of the view. More...
2D camera that defines what region is shown on screen

**sf::View** defines a camera in the 2D scene.

This is a very powerful concept: you can scroll, rotate or zoom the entire scene your drawable objects are drawn.

A view is composed of a source rectangle, which defines what part of the viewport, which defines where the contents of the source rectangle will be displayed (window or texture).

The viewport allows to map the scene to a custom part of the render target, which can be used for split-screen or for displaying a minimap, for example. If the source rectangle is larger than the viewport, its contents will be stretched to fit in.

To apply a view, you have to assign it to the render target. Then, objects affected by the view until you use another view.

Usage example:

```cpp
sf::RenderWindow window;
sf::View view;

// Initialize the view to a rectangle located at (100, 100) and with a size of 400x200
view.reset(sf::FloatRect(100, 100, 400, 200));

// Rotate it by 45 degrees
view.rotate(45);

// Set its target viewport to be half of the window
view.setViewport(sf::FloatRect(0.f, 0.f, 0.5f, 1.f));

// Apply it
window.setView(view);
```

// Render stuff
window.draw(someSprite);

// Set the default view back
window.setView(window.getDefaultView());

// Render stuff not affected by the view
window.draw(someText);

See also the note on coordinates and undistorted rendering in sf::Transformable.

See also

sf::RenderWindow, sf::RenderTexture

Definition at line 43 of file View.hpp.
## Constructor & Destructor Documentation

### sf::View::View ( )

Default constructor.

This constructor creates a default view of (0, 0, 1000, 1000)

### sf::View::View ( const FloatRect & rectangle )

Construct the view from a rectangle.

**Parameters**
- **rectangle** Rectangle defining the zone to display

### sf::View::View ( const Vector2f & center, const Vector2f & size )

Construct the view from its center and size.

**Parameters**
- **center** Center of the zone to display
- **size** Size of zone to display
Member Function Documentation

**const Vector2f& sf::View::getCenter ( ) const**

Get the center of the view.

**Returns**
Center of the view

**See also**
getSize, setCenter

**const Transform& sf::View::getInverseTransform ( ) const**

Get the inverse projection transform of the view.

This function is meant for internal use only.

**Returns**
Inverse of the projection transform defining the view

**See also**
getTransform

**float sf::View::getRotation ( ) const**
Get the current orientation of the view.

**Returns**
Rotation angle of the view, in degrees

**See also**
setRotation

```cpp
const Vector2f& sf::View::getSize () const
```

Get the size of the view.

**Returns**
Size of the view

**See also**
getCenter, setSize

```cpp
const Transform& sf::View::getTransform () const
```

Get the projection transform of the view.

This function is meant for internal use only.

**Returns**
Projection transform defining the view

**See also**
getInverseTransform
**const FloatRect& sf::View::getViewport ( ) const**

Get the target viewport rectangle of the view.

**Returns**
Viewport rectangle, expressed as a factor of the target size

**See also**
setViewport

**void sf::View::move ( float offsetX, float offsetY )**

Move the view relatively to its current position.

**Parameters**
- **offsetX** X coordinate of the move offset
- **offsetY** Y coordinate of the move offset

**See also**
setCenter, rotate, zoom

**void sf::View::move ( const Vector2f & offset )**

Move the view relatively to its current position.

**Parameters**
- **offset** Move offset
See also
   setCenter, rotate, zoom

void sf::View::reset ( const FloatRect & rectangle )

Reset the view to the given rectangle.
Note that this function resets the rotation angle to 0.

Parameters
   rectangle Rectangle defining the zone to display

See also
   setCenter, setSize, setRotation

void sf::View::rotate ( float angle )

Rotate the view relatively to its current orientation.

Parameters
   angle Angle to rotate, in degrees

See also
   setRotation, move, zoom

void sf::View::setCenter ( float x, float y )
Set the center of the view.

**Parameters**
- \( x \) X coordinate of the new center
- \( y \) Y coordinate of the new center

**See also**
- setSize, getCenter

---

```cpp
void sf::View::setCenter ( const Vector2f & center )
```

Set the center of the view.

**Parameters**
- `center` New center

**See also**
- setSize, getCenter

---

```cpp
void sf::View::setRotation ( float angle )
```

Set the orientation of the view.

The default rotation of a view is 0 degree.

**Parameters**
- `angle` New angle, in degrees

**See also**
- getRotation
void sf::View::setSize ( float width, float height )

Set the size of the view.

Parameters

  width  New width of the view
  height New height of the view

See also

  setCenter, getCenter

void sf::View::setSize ( const Vector2f & size )

Set the size of the view.

Parameters

  size New size

See also

  setCenter, getCenter

void sf::View::setViewport ( const FloatRect & viewport )

Set the target viewport.

The viewport is the rectangle into which the contents of the view are clipped (between 0 and 1) of the size of the RenderTarget to which the view is applied.
which takes the left side of the target would be defined with `View.setViewport()`.

By default, a view has a viewport which covers the entire target.

**Parameters**

- **viewport** New viewport rectangle

**See also**

- `getViewport`

---

**void sf::View::zoom ( float factor )**

Resize the view rectangle relatively to its current size.

Resizing the view simulates a zoom, as the zone displayed on screen grows or shrinks.

**Multiplier:**

- 1 keeps the size unchanged
- > 1 makes the view bigger (objects appear smaller)
- < 1 makes the view smaller (objects appear bigger)

**Parameters**

- **factor** Zoom factor to apply

**See also**

- `setSize`, `move`, `rotate`

---

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

- `View.hpp`
Network module

Socket-based communication, utilities and higher-level network protocols.
## Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sf::Ftp</code></td>
<td>A FTP client. More...</td>
</tr>
<tr>
<td><code>sf::Http</code></td>
<td>A HTTP client. More...</td>
</tr>
<tr>
<td><code>sf::IpAddress</code></td>
<td>Encapsulate an IPv4 network address. More...</td>
</tr>
<tr>
<td><code>sf::Packet</code></td>
<td>Utility class to build blocks of data to transfer over the network. More...</td>
</tr>
<tr>
<td><code>sf::Socket</code></td>
<td>Base class for all the socket types. More...</td>
</tr>
<tr>
<td><code>sf::SocketSelector</code></td>
<td>Multiplexer that allows to read from multiple sockets. More...</td>
</tr>
<tr>
<td><code>sf::TcpListener</code></td>
<td>Socket that listens to new TCP connections. More...</td>
</tr>
<tr>
<td><code>sf::TcpSocket</code></td>
<td>Specialized socket using the TCP protocol. More...</td>
</tr>
<tr>
<td><code>sf::UdpSocket</code></td>
<td>Specialized socket using the UDP protocol. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Socket-based communication, utilities and higher-level network protocols

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
A FTP client. More...

#include <Ftp.hpp>

Inheritance diagram for sf::Ftp:
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryResponse</td>
<td>Specialization of FTP response returning a directory. <a href="#">More...</a></td>
</tr>
<tr>
<td>ListingResponse</td>
<td>Specialization of FTP response returning a filename listing. <a href="#">More</a></td>
</tr>
<tr>
<td>Response</td>
<td>Define a FTP response. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Public Types

<table>
<thead>
<tr>
<th>enum</th>
<th>TransferMode { Binary, Ascii, Ebcdic }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enumeration of transfer modes. More...</td>
</tr>
</tbody>
</table>

## Public Member Functions

<table>
<thead>
<tr>
<th>Class</th>
<th>Name</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>~Ftp ()</td>
<td>Destructor.</td>
<td></td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>connect</td>
<td>Connect to the specified FTP server.</td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>disconnect ()</td>
<td>Close the connection with the server.</td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>login ()</td>
<td>Log in using an anonymous account.</td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>login (const std::string &amp;name, const std::string &amp;password)</td>
<td>Log in using a username and a password.</td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>keepAlive ()</td>
<td>Send a null command to keep the connection alive.</td>
<td></td>
</tr>
<tr>
<td>DirectoryResponse</td>
<td>getWorkingDirectory ()</td>
<td>Get the current working directory.</td>
<td>More...</td>
</tr>
<tr>
<td>ListingResponse</td>
<td>getDirectoryListing ()</td>
<td>Get the contents of the given directory.</td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>changeDirectory (const std::string &amp;directory)</td>
<td>Change the current working directory.</td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>parentDirectory ()</td>
<td>Go to the parent directory of the current one.</td>
<td>More...</td>
</tr>
<tr>
<td>Response</td>
<td>Function</td>
<td>Description</td>
<td>More...</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td><code>createDirectory</code></td>
<td>(const std::string &amp;name) Create a new directory.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>deleteDirectory</code></td>
<td>(const std::string &amp;name) Remove an existing directory.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>renameFile</code></td>
<td>(const std::string &amp;file, const std::string &amp;newName) Rename an existing file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>deleteFile</code></td>
<td>(const std::string &amp;name) Remove an existing file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>download</code></td>
<td>(const std::string &amp;remoteFile, const std::string &amp;localPath, mode=Binary)</td>
<td>Download a file from the server.</td>
</tr>
<tr>
<td></td>
<td><code>upload</code></td>
<td>(const std::string &amp;localFile, const std::string &amp;remotePath, mode=Binary)</td>
<td>Upload a file to the server.</td>
</tr>
<tr>
<td></td>
<td><code>sendCommand</code></td>
<td>(const std::string &amp;command, const std::string &amp;parameter) Send a command to the FTP server.</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>DataChannel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Friends
A FTP client.

sf::Ftp is a very simple FTP client that allows you to communicate with a FTP server.

The FTP protocol allows you to manipulate a remote file system (list, remove, ...).

Using the FTP client consists of 4 parts:

- Connecting to the FTP server
- Logging in (either as a registered user or anonymously)
- Sending commands to the server
- Disconnecting (this part can be done implicitly by the destructor)

Every command returns a FTP response, which contains the status code and message from the server. Some commands such as getWorkingDirectory() and getDirectoryListing() and use a class derived from sf::Ftp::Response to provide this data. The most often used commands are directly provided as member functions, but it is also possible to use sendCommand() function.

Note that response statuses >= 1000 are not part of the FTP standard, they are generated by SFML when an internal error occurs.

All commands, especially upload and download, may take some time to complete. Thus you should know if you don’t want to block your application while the server is completing the task.

Usage example:

```cpp
// Create a new FTP client
```
sf::Ftp ftp;

// Connect to the server
if (response.isOk())
    std::cout << "Connected" << std::endl;

// Log in
response = ftp.login("laurent", "dF6Zm89D");
if (response.isOk())
    std::cout << "Logged in" << std::endl;

// Print the working directory
sf::Ftp::DirectoryResponse directory = ftp.getWorkingDirectory();
if (directory.isOk())
    std::cout << "Working directory: " << directory.getDirectory() << std::endl;

// Create a new directory
response = ftp.createDirectory("files");
if (response.isOk())
    std::cout << "Created new directory" << std::endl;

// Upload a file to this new directory
response = ftp.upload("local-path/file.txt", "files", sf::Ftp::Ascii);
if (response.isOk())
    std::cout << "File uploaded" << std::endl;

// Send specific commands (here: FEAT to list supported FTP features)
response = ftp.sendCommand("FEAT");
if (response.isOk())
    std::cout << "Feature list:\n" << response.getMessage() << std::endl;

// Disconnect from the server (optional)
ftp.disconnect();

Definition at line 47 of file Ftp.hpp.
## Member Enumeration Documentation

**enum sf::Ftp::TransferMode**

Enumeration of transfer modes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>Binary mode (file is transfered as a sequence of bytes)</td>
</tr>
<tr>
<td>Ascii</td>
<td>Text mode using ASCII encoding.</td>
</tr>
<tr>
<td>Ebcdic</td>
<td>Text mode using EBCDIC encoding.</td>
</tr>
</tbody>
</table>

Definition at line 55 of file *Ftp.hpp*. 
sf::Ftp::~Ftp ( )

Destructor.

Automatically closes the connection with the server if it is still opened.
**Response** sf::Ftp::changeDirectory ( const std::string & directory )

Change the current working directory.

The new directory must be relative to the current one.

**Parameters**
- **directory** New working directory

**Returns**
- Server response to the request

**See also**

---

**Response** sf::Ftp::connect ( const IpAddress & server,
                              unsigned short port = 21,
                              Time timeout = Time::Zero )

Connect to the specified FTP server.

The port has a default value of 21, which is the standard port used by
FTP. You should use a different value, unless you really know what you do. This function
may take a while to complete, especially if the server is not reachable.
To avoid blocking your application for too long, you can use a timeout. The default value, **Time::Zero**,
timeout will be used (which is usually pretty long).

Parameters

server Name or address of the FTP server to connect to
port Port used for the connection
timeout Maximum time to wait

Returns

Server response to the request

See also
disconnect

Response sf::Ftp::createDirectory ( const std::string & name )

Create a new directory.

The new directory is created as a child of the current working directory.

Parameters

name Name of the directory to create

Returns

Server response to the request

See also
deleteDirectory

Response sf::Ftp::deleteDirectory ( const std::string & name )

Remove an existing directory.
The directory to remove must be relative to the current working directory; the directory will be removed permanently!

**Parameters**
- **name** Name of the directory to remove

**Returns**
- Server response to the request

**See also**
- `createDirectory`

---

**Response** sf::Ftp::deleteFile ( const std::string & name )

Remove an existing file.

The file name must be relative to the current working directory. Use this be removed permanently!

**Parameters**
- **name** File to remove

**Returns**
- Server response to the request

**See also**
- `renameFile`

---

**Response** sf::Ftp::disconnect ( )

Close the connection with the server.
Returns
Server response to the request

See also
connect

Response `sf::Ftp::download` (const `std::string` & `remoteFile`, const `std::string` & `localPath`, `TransferMode` `mode` = `Binary`)

Download a file from the server.

The filename of the distant file is relative to the current working directory of the server, and the local destination path is relative to the current directory of your application. If the distant file already exists in the local destination path, it will be overwritten.

Parameters

- `remoteFile` Filename of the distant file to download
- `localPath` The directory in which to put the file on the local computer
- `mode` Transfer mode

Returns
Server response to the request

See also
upload

ListingResponse `sf::Ftp::getDirectoryListing` (const `std::string` & `directory`)
Get the contents of the given directory.

This function retrieves the sub-directories and files contained in the given directory. The `directory` parameter is relative to the current working directory.

**Parameters**
- `directory` Directory to list

**Returns**
- Server response to the request

**See also**

---

**DirectoryResponse** sf::Ftp::getWorkingDirectory ( )

Get the current working directory.

The working directory is the root path for subsequent operations involving directories and filenames.

**Returns**
- Server response to the request

**See also**
- `getDirectoryListing`, `changeDirectory`, `parentDirectory`

---

**Response** sf::Ftp::keepAlive ( )

Send a null command to keep the connection alive.
This command is useful because the server may close the connection sent.

**Returns**
Server response to the request

### Response sf::Ftp::login ( )

Log in using an anonymous account.

Logging in is mandatory after connecting to the server. Users that are not logged in cannot perform any operation.

**Returns**
Server response to the request

### Response sf::Ftp::login ( const std::string & name, const std::string & password )

Log in using a username and a password.

Logging in is mandatory after connecting to the server. Users that are not logged in cannot perform any operation.

**Parameters**
- name User name
- password Password

**Returns**
Server response to the request
**Response** sf::Ftp::parentDirectory ( )

Go to the parent directory of the current one.

**Returns**

Server response to the request

**See also**

getWorkingDirectory, getDirectoryListing, changeDirectory

---

**Response** sf::Ftp::renameFile ( const std::string & file, const std::string & newName )

Rename an existing file.

The filenames must be relative to the current working directory.

**Parameters**

- **file** File to rename
- **newName** New name of the file

**Returns**

Server response to the request

**See also**

deleteFile

---

**Response** sf::Ftp::sendCommand ( const std::string & command,
Send a command to the FTP server.

While the most often used commands are provided as member functions, you can be used to send any FTP command to the server. If the command requires parameters, they can be specified in `parameter`. If the server returns information, you can extract it from the response using `Response::getMessage()`.

### Parameters

- `command` Command to send
- `parameter` Command parameter

### Returns

Server response to the request

```cpp
Response sf::Ftp::upload (const std::string & localFile,
                        const std::string & remotePath,
                        TransferMode mode = Binary)
```

Upload a file to the server.

The name of the local file is relative to the current working directory of your application, and the remote path is relative to the current directory of the FTP server.

### Parameters

- `localFile` Path of the local file to upload
- `remotePath` The directory in which to put the file on the server
- `mode` Transfer mode
Returns
   Server response to the request

See also
   download

The documentation for this class was generated from the following file:
- Ftp.hpp

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila  ::  Documentation generated by doxygen
sf::Ftp::DirectoryResponse Class Reference

Specialization of FTP response returning a directory. More...

#include <Ftp.hpp>

Inheritance diagram for sf::Ftp::DirectoryResponse:
Public Types

```java
enum Status {
    RestartMarkerReply = 110, ServiceReadySoon = 120, DataConnectionOpeningDataConnection = 150,
    Ok = 200, PointlessCommand = 202, SystemStatus = 211, DirectoryStatus = 213, HelpMessage = 214,
    DataConnectionAlreadyOpened = 221, OpeningDataConnection = 225, ClosingConnection = 227,
    ServiceReadyClosingConnection = 230, DataConnectionOpened = 240, EnteringPassiveMode = 247,
    LoggedIn = 230, FileActionOk = 250, DirectoryOk = 257, NeedPassword = 332,
    NeedAccountToLogIn = 332, NeedInformation = 350, ServiceUnavailable = 425,
    DataConnectionUnavailable = 425, TransferAborted = 426, FileActionAborted = 450, LocalError = 450,
    CommandUnknown = 500, ParametersUnknown = 501, CommandNotImplemented = 503,
    ParameterNotImplemented = 504, NotLoggedln = 530, NeedAccountToStore = 530,
    FileUnavailable = 550, PageTypeUnknown = 551, NotEnoughMemory = 552, FilenameNotAllowed = 552,
    InvalidResponse = 1000, ConnectionFailed = 1001, ConnectionClosed = 1002, InvalidFile = 1002
}
```

Status codes possibly returned by a FTP response. More...
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const std::string &amp; getDirectory () const</code></td>
<td>Get the directory returned in the response.</td>
</tr>
<tr>
<td><code>bool isOk () const</code></td>
<td>Check if the status code means a success.</td>
</tr>
<tr>
<td><code>Status getStatus () const</code></td>
<td>Get the status code of the response.</td>
</tr>
<tr>
<td><code>const std::string &amp; getMessage () const</code></td>
<td>Get the full message contained in the response.</td>
</tr>
</tbody>
</table>
Detailed Description

Specialization of FTP response returning a directory.

Definition at line 188 of file Ftp.hpp.
# Member Enumeration Documentation

**enum sf::Ftp::Response::Status**

Status codes possibly returned by a FTP response.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RestartMarkerReply</td>
<td>Restart marker reply.</td>
</tr>
<tr>
<td>ServiceReadySoon</td>
<td>Service ready in N minutes.</td>
</tr>
<tr>
<td>DataConnectionAlreadyOpened</td>
<td>Data connection already opened, transfer starting.</td>
</tr>
<tr>
<td>OpeningDataConnection</td>
<td>File status ok, about to open data connection.</td>
</tr>
<tr>
<td>Ok</td>
<td>Command ok.</td>
</tr>
<tr>
<td>PointlessCommand</td>
<td>Command not implemented.</td>
</tr>
<tr>
<td>SystemStatus</td>
<td>System status, or system help reply.</td>
</tr>
<tr>
<td>DirectoryStatus</td>
<td>Directory status.</td>
</tr>
<tr>
<td>FileStatus</td>
<td></td>
</tr>
<tr>
<td>Message Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>File status.</td>
<td>File status.</td>
</tr>
<tr>
<td>HelpMessage</td>
<td>Help message.</td>
</tr>
<tr>
<td>SystemType</td>
<td>NAME system type, where NAME is an official system name from the list in the Assigned Numbers document.</td>
</tr>
<tr>
<td>ServiceReady</td>
<td>Service ready for new user.</td>
</tr>
<tr>
<td>ClosingConnection</td>
<td>Service closing control connection.</td>
</tr>
<tr>
<td>DataConnectionOpened</td>
<td>Data connection open, no transfer in progress.</td>
</tr>
<tr>
<td>ClosingDataConnection</td>
<td>Closing data connection, requested file action successful.</td>
</tr>
<tr>
<td>EnteringPassiveMode</td>
<td>Entering passive mode.</td>
</tr>
<tr>
<td>LoggedIn</td>
<td>User logged in, proceed. Logged out if appropriate.</td>
</tr>
<tr>
<td>FileActionOk</td>
<td>Requested file action ok.</td>
</tr>
<tr>
<td>DirectoryOk</td>
<td>PATHNAME created.</td>
</tr>
<tr>
<td>NeedPassword</td>
<td>User name ok, need password.</td>
</tr>
<tr>
<td>NeedAccountToLogIn</td>
<td>Need account for login.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NeedInformation</td>
<td>Requested file action pending further</td>
</tr>
<tr>
<td>ServiceUnavailable</td>
<td>Service not available, closing control</td>
</tr>
<tr>
<td>DataConnectionUnavailable</td>
<td>Can't open data connection.</td>
</tr>
<tr>
<td>TransferAborted</td>
<td>Connection closed, transfer aborted.</td>
</tr>
<tr>
<td>FileActionAborted</td>
<td>Requested file action not taken.</td>
</tr>
<tr>
<td>LocalError</td>
<td>Requested action aborted, local error</td>
</tr>
<tr>
<td>InsufficientStorageSpace</td>
<td>Requested action not taken; insufficient storage space unavailable.</td>
</tr>
<tr>
<td>CommandUnknown</td>
<td>Syntax error, command unrecognized.</td>
</tr>
<tr>
<td>ParametersUnknown</td>
<td>Syntax error in parameters or arguments.</td>
</tr>
<tr>
<td>CommandNotImplemented</td>
<td>Command not implemented.</td>
</tr>
<tr>
<td>BadCommandSequence</td>
<td>Bad sequence of commands.</td>
</tr>
<tr>
<td>ParameterNotImplemented</td>
<td>Command not implemented for that parameter.</td>
</tr>
<tr>
<td>NotLoggedIn</td>
<td>Not logged in.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NeedAccountToStore</td>
<td>Need account for storing files.</td>
</tr>
<tr>
<td>FileUnavailable</td>
<td>Requested action not taken, file unavailable.</td>
</tr>
<tr>
<td>PageTypeUnknown</td>
<td>Requested action aborted, page type unknown.</td>
</tr>
<tr>
<td>NotEnoughMemory</td>
<td>Requested file action aborted, exceeded storage allocation.</td>
</tr>
<tr>
<td>FilenameNotAllowed</td>
<td>Requested action not taken, file name not allowed.</td>
</tr>
<tr>
<td>InvalidResponse</td>
<td>Not part of the FTP standard, generated by SFML when a received response cannot be parsed.</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>Not part of the FTP standard, generated by SFML when the low-level socket connection with the server fails.</td>
</tr>
<tr>
<td>ConnectionClosed</td>
<td>Not part of the FTP standard, generated by SFML when the low-level socket connection is unexpectedly closed.</td>
</tr>
<tr>
<td>InvalidFile</td>
<td>Not part of the FTP standard, generated by SFML when a local file cannot be read or written.</td>
</tr>
</tbody>
</table>

Definition at line 74 of file Ftp.hpp.
sf::Ftp::DirectoryResponse::DirectoryResponse (const Response &)

Default constructor.

**Parameters**

- `response` Source response

Constructor & Destructor Documentation
Member Function Documentation

### const std::string& sf::Ftp::DirectoryResponse::getDirectory() const

Get the directory returned in the response.

**Returns**
- Directory name

### const std::string& sf::Ftp::Response::getMessage() const

Get the full message contained in the response.

**Returns**
- The response message

### Status sf::Ftp::Response::getStatus() const

Get the status code of the response.

**Returns**
- Status code

### bool sf::Ftp::Response::isOk() const
Check if the status code means a success.

This function is defined for convenience, it is equivalent to testing if the status code is less than 400.

**Returns**

True if the status is a success, false if it is a failure

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

- Ftp.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
sf::Ftp::ListingResponse Class Reference

Specialization of FTP response returning a filename listing. More...

#include <Ftp.hpp>

Inheritance diagram for sf::Ftp::ListingResponse:
Public Types

<table>
<thead>
<tr>
<th>Status</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>RestartMarkerReply</td>
<td>110</td>
</tr>
<tr>
<td>ServiceReadySoon</td>
<td>120</td>
</tr>
<tr>
<td>DataConnectionOpening</td>
<td>150</td>
</tr>
<tr>
<td>Ok</td>
<td>200</td>
</tr>
<tr>
<td>PointlessCommand</td>
<td>202</td>
</tr>
<tr>
<td>SystemStatus</td>
<td>211</td>
</tr>
<tr>
<td>DirectoryStatus</td>
<td>213</td>
</tr>
<tr>
<td>HelpMessage</td>
<td>214</td>
</tr>
<tr>
<td>SystemType</td>
<td>215</td>
</tr>
<tr>
<td>FileStatus</td>
<td>213</td>
</tr>
<tr>
<td>FileActionOk</td>
<td>250</td>
</tr>
<tr>
<td>DirectoryOk</td>
<td>257</td>
</tr>
<tr>
<td>NeedPassword</td>
<td>332</td>
</tr>
<tr>
<td>NeedAccountToLogIn</td>
<td>332</td>
</tr>
<tr>
<td>NeedInformation</td>
<td>350</td>
</tr>
<tr>
<td>ServiceUnavailable</td>
<td>425</td>
</tr>
<tr>
<td>DataConnectionUnavailable</td>
<td>425</td>
</tr>
<tr>
<td>TransferAborted</td>
<td>426</td>
</tr>
<tr>
<td>FileActionAborted</td>
<td>450</td>
</tr>
<tr>
<td>LocalError</td>
<td>451</td>
</tr>
<tr>
<td>CommandUnknown</td>
<td>500</td>
</tr>
<tr>
<td>ParametersUnknown</td>
<td>501</td>
</tr>
<tr>
<td>BadCommandSequence</td>
<td>503</td>
</tr>
<tr>
<td>ParameterNotImplemented</td>
<td>504</td>
</tr>
<tr>
<td>NotLoggedin</td>
<td>530</td>
</tr>
<tr>
<td>NeedAccountToStore</td>
<td>632</td>
</tr>
<tr>
<td>FileUnavailable</td>
<td>550</td>
</tr>
<tr>
<td>PageTypeUnknown</td>
<td>551</td>
</tr>
<tr>
<td>NotEnoughMemory</td>
<td>552</td>
</tr>
<tr>
<td>FilenameNotAllowed</td>
<td>552</td>
</tr>
<tr>
<td>InvalidResponse</td>
<td>1000</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>1001</td>
</tr>
<tr>
<td>ConnectionClosed</td>
<td>1002</td>
</tr>
<tr>
<td>InvalidFile</td>
<td>1002</td>
</tr>
</tbody>
</table>

Status codes possibly returned by a FTP response. More...
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListingResponse (const Response &amp;response, const std::string &amp;data)</td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td>const std::vector<a href="">std::string</a> &amp; getListing() const</td>
<td>Return the array of directory/file names</td>
</tr>
<tr>
<td>bool isOk() const</td>
<td>Check if the status code means a success</td>
</tr>
<tr>
<td>Status getStatus() const</td>
<td>Get the status code of the response. More...</td>
</tr>
<tr>
<td>const std::string &amp; getMessage() const</td>
<td>Get the full message contained in the response</td>
</tr>
</tbody>
</table>
Detailed Description

Specialization of FTP response returning a filename listing.

Definition at line 221 of file Ftp.hpp.
### enum sf::Ftp::Response::Status

Status codes possibly returned by a FTP response.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RestartMarkerReply</td>
<td>Restart marker reply.</td>
</tr>
<tr>
<td>ServiceReadySoon</td>
<td>Service ready in N minutes.</td>
</tr>
<tr>
<td>DataConnectionAlreadyOpened</td>
<td>Data connection already opened, transfer starting.</td>
</tr>
<tr>
<td>OpeningDataConnection</td>
<td>File status ok, about to open data connection.</td>
</tr>
<tr>
<td>Ok</td>
<td>Command ok.</td>
</tr>
<tr>
<td>PointlessCommand</td>
<td>Command not implemented.</td>
</tr>
<tr>
<td>SystemStatus</td>
<td>System status, or system help reply.</td>
</tr>
<tr>
<td>DirectoryStatus</td>
<td>Directory status.</td>
</tr>
<tr>
<td>FileStatus</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>File status.</td>
<td></td>
</tr>
<tr>
<td>HelpMessage</td>
<td>Help message.</td>
</tr>
<tr>
<td>SystemType</td>
<td>NAME system type, where NAME is an official system name from the list in the Assigned Numbers document.</td>
</tr>
<tr>
<td>ServiceReady</td>
<td>Service ready for new user.</td>
</tr>
<tr>
<td>ClosingConnection</td>
<td>Service closing control connection.</td>
</tr>
<tr>
<td>DataConnectionOpened</td>
<td>Data connection open, no transfer in progress.</td>
</tr>
<tr>
<td>ClosingDataConnection</td>
<td>Closing data connection, requested file action successful.</td>
</tr>
<tr>
<td>EnteringPassiveMode</td>
<td>Entering passive mode.</td>
</tr>
<tr>
<td>LoggedIn</td>
<td>User logged in, proceed. Logged out if appropriate.</td>
</tr>
<tr>
<td>FileActionOk</td>
<td>Requested file action ok.</td>
</tr>
<tr>
<td>DirectoryOk</td>
<td>PATHNAME created.</td>
</tr>
<tr>
<td>NeedPassword</td>
<td>User name ok, need password.</td>
</tr>
<tr>
<td>NeedAccountToLogIn</td>
<td>Need account for login.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NeedInformation</td>
<td>Requested file action pending further</td>
</tr>
<tr>
<td>ServiceUnavailable</td>
<td>Service not available, closing control</td>
</tr>
<tr>
<td>DataConnectionUnavailable</td>
<td>Can't open data connection.</td>
</tr>
<tr>
<td>TransferAborted</td>
<td>Connection closed, transfer aborted.</td>
</tr>
<tr>
<td>FileActionAborted</td>
<td>Requested file action not taken.</td>
</tr>
<tr>
<td>LocalError</td>
<td>Requested action aborted, local error</td>
</tr>
<tr>
<td>InsufficientStorageSpace</td>
<td>Requested action not taken; insufficient storage unavailable.</td>
</tr>
<tr>
<td>CommandUnknown</td>
<td>Syntax error, command unrecognized</td>
</tr>
<tr>
<td>ParametersUnknown</td>
<td>Syntax error in parameters or arguments</td>
</tr>
<tr>
<td>CommandNotImplemented</td>
<td>Command not implemented.</td>
</tr>
<tr>
<td>BadCommandSequence</td>
<td>Bad sequence of commands.</td>
</tr>
<tr>
<td>ParameterNotImplemented</td>
<td>Command not implemented for that parameter</td>
</tr>
<tr>
<td>NotLoggedIn</td>
<td>Not logged in.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>NeedAccountToStore</td>
<td>Need account for storing files.</td>
</tr>
<tr>
<td>FileUnavailable</td>
<td>Requested action not taken, file unavailable.</td>
</tr>
<tr>
<td>PageTypeUnknown</td>
<td>Requested action aborted, page type unknown.</td>
</tr>
<tr>
<td>NotEnoughMemory</td>
<td>Requested file action aborted, exceeded storage allocation.</td>
</tr>
<tr>
<td>FilenameNotAllowed</td>
<td>Requested action not taken, file name not allowed.</td>
</tr>
<tr>
<td>InvalidResponse</td>
<td>Not part of the FTP standard, generated by SFML when a response cannot be parsed.</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>Not part of the FTP standard, generated by SFML when the low-level socket connection with the server fails.</td>
</tr>
<tr>
<td>ConnectionClosed</td>
<td>Not part of the FTP standard, generated by SFML when the low-level socket connection is unexpectedly closed.</td>
</tr>
<tr>
<td>InvalidFile</td>
<td>Not part of the FTP standard, generated by SFML when a local file cannot be read or written.</td>
</tr>
</tbody>
</table>

Definition at line 74 of file Ftp.hpp.
Constructor & Destructor Documentation

sf::Ftp::ListingResponse::ListingResponse ( const Response & response, const std::string & data )

Default constructor.

Parameters

- **response** Source response
- **data** Data containing the raw listing
### Member Function Documentation

<table>
<thead>
<tr>
<th>Function Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `const std::vector<std::string>& sf::Ftp::ListingResponse::getListing` | Return the array of directory/file names.  
**Returns**  
Array containing the requested listing |
| `const std::string& sf::Ftp::Response::getMessage()` | Get the full message contained in the response.  
**Returns**  
The response message |
| `Status sf::Ftp::Response::getStatus()` | Get the status code of the response.  
**Returns**  
Status code |
| `bool sf::Ftp::Response::isOk()` | |
Check if the status code means a success.

This function is defined for convenience, it is equivalent to testing if the status code is less than 400.

**Returns**

True if the status is a success, false if it is a failure

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

- Ftp.hpp
Define a FTP response. More...

#include <Ftp.hpp>

Inheritance diagram for sf::Ftp::Response:
Public Types

```plaintext
enum Status {  
  RestartMarkerReply = 110,  
  ServiceReadySoon = 120,  
  DataConnectionOpening = 130,  
  OpeningDataConnection = 150,  
  Ok = 200,  
  PointlessCommand = 202,  
  SystemStatus = 211,  
  DirectoryStatus = 212,  
  FileStatus = 213,  
  HelpMessage = 214,  
  SystemType = 215,  
  DirectoryOk = 257,  
  FileActionOk = 250,  
  FileActionAborted = 450,  
  LocalError = 451,  
  TransferAborted = 426,  
  DataConnectionUnavailable = 425,  
  DataConnectionOpened = 225,  
  ClosingConnection = 221,  
  ClosingDataConnection = 227,  
  EnteringPassiveMode = 227,  
  LoggedIn = 230,  
  NeedPassword = 452,  
  NeedAccountToLogIn = 332,  
  NeedInformation = 350,  
  ServiceUnavailable = 425,  
  DataConnectionUnavailable = 425,  
  ServiceReady = 203,  
  FileActionAborted = 450,  
  LocalError = 451,  
  CommandUnknown = 500,  
  ParametersUnknown = 501,  
  CommandNotImplemented = 503,  
  ParameterNotImplemented = 504,  
  NotLoggedIn = 530,  
  NeedAccountToStore = 332,  
  NeedAccountToLogIn = 332,  
  PageTypeUnknown = 551,  
  NotEnoughMemory = 552,  
  FilenameNotAllowed = 553,  
  InvalidResponse = 1000,  
  ConnectionFailed = 1001,  
  ConnectionClosed = 1002,  
  InvalidFile = 502  
}  
```

Status codes possibly returned by a FTP response. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response (Status code=InvalidResponse, const std::string&amp;)</td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool &amp;isOk() const</td>
<td>Check if the status code means a success. <a href="#">More...</a></td>
</tr>
<tr>
<td>Status &amp;getStatus() const</td>
<td>Get the status code of the response. <a href="#">More...</a></td>
</tr>
<tr>
<td>const std::string &amp; &amp;getMessage() const</td>
<td>Get the full message contained in the response. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Define a FTP response.

Definition at line 66 of file Ftp.hpp.
Member Enumeration Documentation

**enum sf::Ftp::Response::Status**

Status codes possibly returned by a FTP response.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RestartMarkerReply</td>
<td>Restart marker reply.</td>
</tr>
<tr>
<td>ServiceReadySoon</td>
<td>Service ready in N minutes.</td>
</tr>
<tr>
<td>DataConnectionAlreadyOpened</td>
<td>Data connection already opened, transfer starting.</td>
</tr>
<tr>
<td>OpeningDataConnection</td>
<td>File status ok, about to open data connection.</td>
</tr>
<tr>
<td>Ok</td>
<td>Command ok.</td>
</tr>
<tr>
<td>PointlessCommand</td>
<td>Command not implemented.</td>
</tr>
<tr>
<td>SystemStatus</td>
<td>System status, or system help reply.</td>
</tr>
<tr>
<td>DirectoryStatus</td>
<td>Directory status.</td>
</tr>
<tr>
<td>FileStatus</td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Message Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File status.</td>
<td>File status.</td>
</tr>
<tr>
<td>HelpMessage</td>
<td>Help message.</td>
</tr>
<tr>
<td>SystemType</td>
<td>NAME system type, where NAME is an official system name from the list in the Assigned Numbers document.</td>
</tr>
<tr>
<td>ServiceReady</td>
<td>Service ready for new user.</td>
</tr>
<tr>
<td>ClosingConnection</td>
<td>Service closing control connection.</td>
</tr>
<tr>
<td>DataConnectionOpened</td>
<td>Data connection open, no transfer in progress.</td>
</tr>
<tr>
<td>ClosingDataConnection</td>
<td>Closing data connection, requested file action successful.</td>
</tr>
<tr>
<td>EnteringPassiveMode</td>
<td>Entering passive mode.</td>
</tr>
<tr>
<td>LoggedIn</td>
<td>User logged in, proceed. Logged out if appropriate.</td>
</tr>
<tr>
<td>FileActionOk</td>
<td>Requested file action ok.</td>
</tr>
<tr>
<td>DirectoryOk</td>
<td>PATHNAME created.</td>
</tr>
<tr>
<td>NeedPassword</td>
<td>User name ok, need password.</td>
</tr>
<tr>
<td>NeedAccountToLogIn</td>
<td>Need account for login.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>NeedInformation</td>
<td>Requested file action pending further</td>
</tr>
<tr>
<td>ServiceUnavailable</td>
<td>Service not available, closing control</td>
</tr>
<tr>
<td>DataConnectionUnavailable</td>
<td>Can't open data connection.</td>
</tr>
<tr>
<td>TransferAborted</td>
<td>Connection closed, transfer aborted.</td>
</tr>
<tr>
<td>FileActionAborted</td>
<td>Requested file action not taken.</td>
</tr>
<tr>
<td>LocalError</td>
<td>Requested action aborted, local error</td>
</tr>
<tr>
<td>InsufficientStorageSpace</td>
<td>Requested action not taken; insufficient space</td>
</tr>
<tr>
<td>CommandUnknown</td>
<td>Syntax error, command unrecognized</td>
</tr>
<tr>
<td>ParametersUnknown</td>
<td>Syntax error in parameters or arguments</td>
</tr>
<tr>
<td>CommandNotImplemented</td>
<td>Command not implemented.</td>
</tr>
<tr>
<td>BadCommandSequence</td>
<td>Bad sequence of commands.</td>
</tr>
<tr>
<td>ParameterNotImplemented</td>
<td>Command not implemented for that parameter</td>
</tr>
<tr>
<td>NotLoggedIn</td>
<td>Not logged in.</td>
</tr>
<tr>
<td>Error Category</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>NeedAccountToStore</td>
<td>Need account for storing files.</td>
</tr>
<tr>
<td>FileUnavailable</td>
<td>Requested action not taken, file unavailable.</td>
</tr>
<tr>
<td>PageTypeUnknown</td>
<td>Requested action aborted, page type unknown.</td>
</tr>
<tr>
<td>NotEnoughMemory</td>
<td>Requested file action aborted, exceeded storage.</td>
</tr>
<tr>
<td>FilenameNotAllowed</td>
<td>Requested action not taken, file name not allowed.</td>
</tr>
<tr>
<td>InvalidResponse</td>
<td>Not part of the FTP standard, generated response cannot be parsed.</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>Not part of the FTP standard, generated low level socket connection with the server failed.</td>
</tr>
<tr>
<td>ConnectionClosed</td>
<td>Not part of the FTP standard, generated low level socket connection is unexpectedly closed.</td>
</tr>
<tr>
<td>InvalidFile</td>
<td>Not part of the FTP standard, generated local file cannot be read or written.</td>
</tr>
</tbody>
</table>

Definition at line 74 of file Ftp.hpp.
sf::Ftp::Response::Response ( Status code = InvalidResponse, const std::string & message = ""
)

Default constructor.

This constructor is used by the FTP client to build the response.

**Parameters**

- `code` (Response status code)
- `message` (Response message)
## Member Function Documentation

### `const std::string& sf::Ftp::Response::getMessage () const`

Get the full message contained in the response.

**Returns**
- The response message

### `Status sf::Ftp::Response::getStatus () const`

Get the status code of the response.

**Returns**
- Status code

### `bool sf::Ftp::Response::isOk () const`

Check if the status code means a success.

This function is defined for convenience, it is equivalent to testing if the status code is less than 400.

**Returns**
- True if the status is a success, false if it is a failure

The documentation for this class was generated from the following file:
sf::Http Class Reference
Network module

A HTTP client. More...

#include <Http.hpp>

Inheritance diagram for sf::Http:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Define a HTTP request. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Define a HTTP response. More...</td>
</tr>
</tbody>
</table>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Http()</code></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td><code>Http(const std::string &amp;host, unsigned short port=0)</code></td>
<td>Construct the HTTP client with the target host. More...</td>
</tr>
<tr>
<td><code>void setHost(const std::string &amp;host, unsigned short port=0)</code></td>
<td>Set the target host. More...</td>
</tr>
<tr>
<td><code>Response sendRequest(const Request &amp;request, Time timeout=Time::Zero)</code></td>
<td>Send a HTTP request and return the server's response. More...</td>
</tr>
</tbody>
</table>
Detailed Description

A HTTP client.

`sf::Http` is a very simple HTTP client that allows you to communicate with a web server.

You can retrieve web pages, send data to an interactive resource, download a remote file, etc. The HTTPS protocol is not supported.

The HTTP client is split into 3 classes:

- `sf::Http::Request`
- `sf::Http::Response`
- `sf::Http`

`sf::Http::Request` builds the request that will be sent to the server. A request is made of:
  - a method (what you want to do)
  - a target URI (usually the name of the web page or file)
  - one or more header fields (options that you can pass to the server)
  - an optional body (for POST requests)

`sf::Http::Response` parses the response from the web server and provides getters to read them. The response contains:
  - a status code
  - header fields (that may be answers to the ones that you requested)
  - a body, which contains the contents of the requested resource

`sf::Http` provides a simple function, `SendRequest`, to send a `sf::Http::Request`
sf::Http::Response from the server.

Usage example:

```cpp
// Create a new HTTP client
sf::Http http;

// We'll work on http://www.sfml-dev.org
http.setHost("http://www.sfml-dev.org");

// Prepare a request to get the 'features.php' page
sf::Http::Request request("features.php");

// Send the request
sf::Http::Response response = http.sendRequest(request);

// Check the status code and display the result
sf::Http::Response::Status status = response.getStatus();
if (status == sf::Http::Response::Ok)
{
    std::cout << response.getBody() << std::endl;
}
else
{
    std::cout << "Error " << status << std::endl;
}
```

Definition at line 46 of file Http.hpp.
Constructor & Destructor Documentation

sf::Http::Http ( )

Default constructor.

sf::Http::Http ( const std::string & host,
               unsigned short port = 0 )

Construct the HTTP client with the target host.

This is equivalent to calling setHost(host, port). The port has a default \ndefault of 0, so the HTTP client will use the right port according to the protocol used (80 for \nHTTP) unless you really need a port other than the standard one, or use an unknown protocol.

Parameters

host Web server to connect to
port Port to use for connection
Member Function Documentation

**Response** sf::Http::sendRequest ( const Request & request, Time timeout = Time::Zero )

Send a HTTP request and return the server's response.

You must have a valid host before sending a request (see setHost). Any in the request will be added with an appropriate value. Warning: this response and may not return instantly; use a thread if you don't want to timeout to limit the time to wait. A value of Time::Zero means that the timeout (which is usually pretty long).

**Parameters**
- request Request to send
- timeout Maximum time to wait

**Returns**
Server's response

**void sf::Http::setHost ( const std::string & host, unsigned short port = 0 )**

Set the target host.

This function just stores the host address and port, it doesn't actually
request. The port has a default value of 0, which means that the HTTP port will be determined according to the protocol used (80 for HTTP). You should leave it like this unless you really need a port other than the standard one, or use an unknown protocol.

**Parameters**

- **host** Web server to connect to
- **port** Port to use for connection

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

- Http.hpp
sf::Http::Request Class Reference

Define a HTTP request. More...

#include <Http.hpp>
Public Types

```csharp
enum Method {
    Get, Post, Head, Put,
    Delete
}

Enumerate the available HTTP methods for a request. More...
```
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Request</code></td>
<td>(const std::string &amp;uri=&quot;/&quot;, Method method=Get, const std::string &amp;body=&quot;&quot;) Default constructor. More...</td>
</tr>
<tr>
<td><code>void setField</code></td>
<td>(const std::string &amp;field, const std::string &amp;value) Set the value of a field. More...</td>
</tr>
<tr>
<td><code>void setMethod</code></td>
<td>(Method method) Set the request method. More...</td>
</tr>
<tr>
<td><code>void setUri</code></td>
<td>(const std::string &amp;uri) Set the requested URI. More...</td>
</tr>
<tr>
<td><code>void setHttpVersion</code></td>
<td>(unsigned int major, unsigned int minor) Set the HTTP version for the request. More...</td>
</tr>
<tr>
<td><code>void setBody</code></td>
<td>(const std::string &amp;body) Set the body of the request. More...</td>
</tr>
</tbody>
</table>
Friends

```java
class HTTP
```
Define a HTTP request.

Definition at line 54 of file Http.hpp.
**Member Enumeration Documentation**

### `enum sf::Http::Request::Method`

Enumerate the available HTTP methods for a request.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Request in get mode, standard method to retrieve a page.</td>
</tr>
<tr>
<td>Post</td>
<td>Request in post mode, usually to send data to a page.</td>
</tr>
<tr>
<td>Head</td>
<td>Request a page's header only.</td>
</tr>
<tr>
<td>Put</td>
<td>Request in put mode, useful for a REST API.</td>
</tr>
<tr>
<td>Delete</td>
<td>Request in delete mode, useful for a REST API.</td>
</tr>
</tbody>
</table>

Definition at line 62 of file Http.hpp.
sf::Http::Request::Request ( const std::string & uri = "/", Method method = get, const std::string & body = "" )

Default constructor.

This constructor creates a GET request, with the root URI ("/") and an empty body.

**Parameters**

- **uri**  
  Target URI
- **method**  
  Method to use for the request
- **body**  
  Content of the request's body
void sf::Http::Request::setBody ( const std::string & body )

Set the body of the request.

The body of a request is optional and only makes sense for POST request methods. The body is empty by default.

**Parameters**

- **body** Content of the body

void sf::Http::Request::setField ( const std::string & field, const std::string & value )

Set the value of a field.

The field is created if it doesn't exist. The name of the field is case-insensitive. By default, a request doesn't contain any field (but the mandatory fields are added later by the HTTP client when sending the request).

**Parameters**

- **field** Name of the field to set
- **value** Value of the field
void sf::Http::Request::setHttpVersion (unsigned int major, unsigned int minor)

Set the HTTP version for the request.

The HTTP version is 1.0 by default.

**Parameters**

- major Major HTTP version number
- minor Minor HTTP version number

void sf::Http::Request::setMethod (Method method)

Set the request method.

See the Method enumeration for a complete list of all the available methods.

Http::Request::Get by default.

**Parameters**

- method Method to use for the request

void sf::Http::Request::setUri (const std::string & uri)

Set the requested URI.

The URI is the resource (usually a web page or a file) that you want to access.

root page) by default.

**Parameters**

- uri URI to request, relative to the host
The documentation for this class was generated from the following file:
- Http.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
Define a HTTP response. More...

#include <Http.hpp>
## Public Types

<table>
<thead>
<tr>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ok = 200, Created = 201, Accepted = 202, NoContent = 204, ResetContent = 205, PartialContent = 206, MultipleChoices = 300, MovedTemporarily = 302, NotModified = 304, BadRequest = 400, Forbidden = 403, NotFound = 404, RangeNotSatisfiable = 407, NotImplemented = 501, BadGateway = 502, ServiceUnavailable = 504, VersionNotSupported = 505, InvalidResponse = 1000, Connect = 101</td>
</tr>
</tbody>
</table>

Enumerate all the valid status codes for a response. [More...](#)
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response ()</td>
<td>Default constructor.</td>
<td></td>
</tr>
<tr>
<td>const std::string &amp;</td>
<td>getField (const std::string &amp;field) const</td>
<td></td>
</tr>
<tr>
<td>Status getStatus () const</td>
<td>Get the response status code.</td>
<td></td>
</tr>
<tr>
<td>unsigned int getMajorHttpVersion () const</td>
<td>Get the major HTTP version number of the response.</td>
<td></td>
</tr>
<tr>
<td>unsigned int getMinorHttpVersion () const</td>
<td>Get the minor HTTP version number of the response.</td>
<td></td>
</tr>
<tr>
<td>const std::string &amp;</td>
<td>getBody () const</td>
<td></td>
</tr>
</tbody>
</table>
Friends

| class | Http |
Detailed Description

Define a HTTP response.

Definition at line 193 of file Http.hpp.
### enum sf::Http::Response::Status

Enumerate all the valid status codes for a response.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ok</td>
<td>Most common code returned when operation was successful.</td>
</tr>
<tr>
<td>Created</td>
<td>The resource has successfully been created.</td>
</tr>
<tr>
<td>Accepted</td>
<td>The request has been accepted, but will be processed later by the server.</td>
</tr>
<tr>
<td>NoContent</td>
<td>The server didn't send any data in return.</td>
</tr>
<tr>
<td>ResetContent</td>
<td>The server informs the client that it should clear the view that caused the request to be sent.</td>
</tr>
<tr>
<td>PartialContent</td>
<td>The server has sent a part of the resource, a response to a partial GET request.</td>
</tr>
<tr>
<td>MultipleChoices</td>
<td>The requested page can be accessed from several locations.</td>
</tr>
<tr>
<td>MovedPermanently</td>
<td>The requested page has permanently moved to another location.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MovedTemporarily</td>
<td>The requested page has temporarily moved to a new location.</td>
</tr>
<tr>
<td>NotModified</td>
<td>For conditional requests, means the requested page hasn't changed and doesn't need to be refreshed.</td>
</tr>
<tr>
<td>BadRequest</td>
<td>The server couldn't understand the request (syntax error).</td>
</tr>
<tr>
<td>Unauthorized</td>
<td>The requested page needs an authentication to be accessed.</td>
</tr>
<tr>
<td>Forbidden</td>
<td>The requested page cannot be accessed at all, even with authentication.</td>
</tr>
<tr>
<td>NotFound</td>
<td>The requested page doesn't exist.</td>
</tr>
<tr>
<td>RangeNotSatisfiable</td>
<td>The server can't satisfy the partial GET request (with a &quot;Range&quot; header field).</td>
</tr>
<tr>
<td>InternalServerError</td>
<td>The server encountered an unexpected error.</td>
</tr>
<tr>
<td>NotImplemented</td>
<td>The server doesn't implement a requested feature.</td>
</tr>
<tr>
<td>BadGateway</td>
<td>The gateway server has received an error from the source server.</td>
</tr>
<tr>
<td>ServiceNotAvailable</td>
<td>The server is temporarily unavailable (overloaded, in maintenance, ...)</td>
</tr>
<tr>
<td>GatewayTimeout</td>
<td>The gateway server couldn't receive a response from the source server.</td>
</tr>
<tr>
<td>VersionNotSupported</td>
<td>The server doesn't support the requested HTTP version.</td>
</tr>
<tr>
<td>InvalidResponse</td>
<td>Response is not a valid HTTP one.</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>Connection with server failed.</td>
</tr>
</tbody>
</table>

Definition at line 201 of file Http.hpp.
Constructor & Destructor Documentation

sf::Http::Response::Response ( )

Default constructor.

Constructs an empty response.
**Member Function Documentation**

---

```cpp
const std::string& sf::Http::Response::getBody () const
```

Get the body of the response.

The body of a response may contain:

- the requested page (for GET requests)
- a response from the server (for POST requests)
- nothing (for HEAD requests)
- an error message (in case of an error)

**Returns**

The response body

---

```cpp
const std::string& sf::Http::Response::getField ( const std::string & field )
```

Get the value of a field.

If the field `field` is not found in the response header, the empty string is returned. This function uses case-insensitive comparisons.

**Parameters**

- `field` Name of the field to get

**Returns**
Value of the field, or empty string if not found

```cpp
unsigned int sf::Http::Response::getMajorHttpVersion ( ) const
```

Get the major HTTP version number of the response.

**Returns**
Major HTTP version number

**See also**
getMinorHttpVersion

```cpp
unsigned int sf::Http::Response::getMinorHttpVersion ( ) const
```

Get the minor HTTP version number of the response.

**Returns**
Minor HTTP version number

**See also**
getMajorHttpVersion

```cpp
Status sf::Http::Response::getStatus ( ) const
```

Get the response status code.

The status code should be the first thing to be checked after receiving a response, a failure or anything else (see the Status enumeration).

**Returns**
Status code of the response

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

- Http.hpp

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
Encapsulate an IPv4 network address. More...

#include <IpAddress.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IpAddress ()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IpAddress (const std::string &amp;address)</code></td>
<td>Construct the address from a string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IpAddress (const char *address)</code></td>
<td>Construct the address from a string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IpAddress (Uint8 byte0, Uint8 byte1, Uint8 byte2, Uint8 byte3)</code></td>
<td>Construct the address from 4 bytes. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IpAddress (Uint32 address)</code></td>
<td>Construct the address from a 32-bits integer. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>std::string toString () const</code></td>
<td>Get a string representation of the address. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>Uint32 toInteger () const</code></td>
<td>Get an integer representation of the address. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>static IpAddress getLocalAddress()</td>
<td>Get the computer's local address.</td>
<td>More...</td>
</tr>
<tr>
<td>static IpAddress getPublicAddress(Time timeout=Time::Zero)</td>
<td>Get the computer's public address.</td>
<td>More...</td>
</tr>
</tbody>
</table>
# Static Public Attributes

<table>
<thead>
<tr>
<th>static const ipAddress</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
<td>Value representing an empty/invalid address. More...</td>
</tr>
<tr>
<td>Any</td>
<td>Any</td>
<td>Value representing any address (0.0.0.0) More...</td>
</tr>
<tr>
<td>LocalHost</td>
<td>LocalHost</td>
<td>The &quot;localhost&quot; address (for connecting a computer to itself locally)</td>
</tr>
<tr>
<td>Broadcast</td>
<td>Broadcast</td>
<td>The &quot;broadcast&quot; address (for sending UDP messages to everyone on a local network) More...</td>
</tr>
</tbody>
</table>
bool operator< (const IpAddress &left, const IpAddress &right)
Overload of < operator to compare two IP addresses. More...
Detailed Description

Encapsulate an IPv4 network address.

`sf::IpAddress` is a utility class for manipulating network addresses.

It provides a set a implicit constructors and conversion functions to easily from/to various representations.

Usage example:

```cpp
sf::IpAddress a0; // an invalid address
sf::IpAddress a1 = sf::IpAddress::None; // an invalid address
sf::IpAddress a2("127.0.0.1"); // the local host
sf::IpAddress a3 = sf::IpAddress::Broadcast; // the broadcast
sf::IpAddress a4(192, 168, 1, 56); // a local address
sf::IpAddress a5("my_computer"); // a local address
sf::IpAddress a6("89.54.1.169"); // a distant address
sf::IpAddress a7("www.google.com"); // a distant address
sf::IpAddress a8 = sf::IpAddress::getLocalAddress(); // my address on
sf::IpAddress a9 = sf::IpAddress::getPublicAddress(); // my address on
```

Note that `sf::IpAddress` currently doesn't support IPv6 nor other types of networks.

Definition at line 44 of file `IpAddress.hpp`.

Constructor & Destructor Documentation

sf::IpAddress::IpAddress ( )

Default constructor.

This constructor creates an empty (invalid) address

sf::IpAddress::IpAddress ( const std::string & address )

Construct the address from a string.

Here address can be either a decimal address (ex: "192.168.1.56") or a network name.

Parameters

address IP address or network name

sf::IpAddress::IpAddress ( const char * address )

Construct the address from a string.

Here address can be either a decimal address (ex: "192.168.1.56") or a network name.

This is equivalent to the constructor taking a std::string parameter, it is defined for convenience so that the implicit conversions from literal strings to IpAddress work.

Parameters
address IP address or network name

sf::IpAddress::IpAddress (Uint8 byte0, Uint8 byte1, Uint8 byte2, Uint8 byte3)

Construct the address from 4 bytes.

Calling IpAddress(a, b, c, d) is equivalent to calling IpAddress("a.b.c.d") parse a string to get the address components.

Parameters
  byte0 First byte of the address
  byte1 Second byte of the address
  byte2 Third byte of the address
  byte3 Fourth byte of the address

sf::IpAddress::IpAddress (Uint32 address)

Construct the address from a 32-bits integer.

This constructor uses the internal representation of the address for optimization purposes, and only if you got that representation from IpAd

Parameters
  address 4 bytes of the address packed into a 32-bits integer

See also
toInteger
Member Function Documentation

**static IpAddress sf::IpAddress::getLocalAddress ( )**

Get the computer's local address.

The local address is the address of the computer from the LAN point of view, i.e., something like 192.168.1.56. It is meaningful only for communications over the local network. Unlike `getPublicAddress`, this function is fast and may be used safely anywhere.

**Returns**
- Local IP address of the computer

**See also**
- `getPublicAddress`

**static IpAddress sf::IpAddress::getPublicAddress ( Time timeout =**

Get the computer's public address.

The public address is the address of the computer from the internet point of view, i.e., something like 89.54.1.169. It is necessary for communications over the world wide web. The only way to get a public address is to ask it to a distant website; as a consequence, this function depends on the network connection and the server, and may be very slow. You should use it as a last resort. You may use a timeout if you don't want your program to be possibly stuck waiting in case there is a problem; this limit is deactivated by default.
### Parameters

- **timeout** Maximum time to wait

### Returns

- Public IP address of the computer

### See also

- `getLocalAddress`

---

### `Uint32 sf::IpAddress::toInteger ( ) const`

Get an integer representation of the address.

The returned number is the internal representation of the address, and purposes only (like sending the address through a socket). The integer then be converted back to a `sf::IpAddress` with the proper constructor.

### Returns

- 32-bits unsigned integer representation of the address

### See also

- `toString`

---

### `std::string sf::IpAddress::toString ( ) const`

Get a string representation of the address.

The returned string is the decimal representation of the IP address (like constructed from a host name).

### Returns
String representation of the address

See also
toInteger
Friends And Related Function Documentation

```cpp
bool operator< ( const IpAddress & left,
                const IpAddress & right
 )
```

Overload of < operator to compare two IP addresses.

**Parameters**

- **left**  
  Left operand (a IP address)

- **right**  
  Right operand (a IP address)

**Returns**

True if *left* is lesser than *right*
## Member Data Documentation

### `const IpAddress sf::IpAddress::Any`

Value representing any address (0.0.0.0)

Definition at line 185 of file IpAddress.hpp.

### `const IpAddress sf::IpAddress::Broadcast`

The "broadcast" address (for sending UDP messages to everyone on a local network)

Definition at line 187 of file IpAddress.hpp.

### `const IpAddress sf::IpAddress::LocalHost`

The "localhost" address (for connecting a computer to itself locally)

Definition at line 186 of file IpAddress.hpp.

### `const IpAddress sf::IpAddress::None`

Value representing an empty/invalid address.

Definition at line 184 of file IpAddress.hpp.
The documentation for this class was generated from the following file:

- IpAddress.hpp

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila. :: Documentation generated by doxygen
Utility class to build blocks of data to transfer over the network. More...

```
#include <Packet.hpp>
```
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Packet()</code></td>
<td>Default constructor.</td>
<td></td>
</tr>
<tr>
<td><code>virtual ~Packet()</code></td>
<td>Virtual destructor.</td>
<td></td>
</tr>
<tr>
<td><code>void append (const void *data, std::size_t sizeInBytes)</code></td>
<td>Append data to the end of the packet.</td>
<td></td>
</tr>
<tr>
<td><code>void clear()</code></td>
<td>Clear the packet.</td>
<td></td>
</tr>
<tr>
<td><code>const void * getData () const</code></td>
<td>Get a pointer to the data contained in the packet.</td>
<td></td>
</tr>
<tr>
<td><code>std::size_t getDataSize () const</code></td>
<td>Get the size of the data contained in the packet.</td>
<td></td>
</tr>
<tr>
<td><code>bool endOfPacket () const</code></td>
<td>Tell if the reading position has reached the end of the packet.</td>
<td></td>
</tr>
<tr>
<td><code>operator BoolType () const</code></td>
<td>Test the validity of the packet, for reading.</td>
<td></td>
</tr>
<tr>
<td><code>Packet &amp; operator&gt;&gt;(bool &amp;data)</code></td>
<td>Overloads of operator &gt;&gt; to read data from the packet.</td>
<td></td>
</tr>
<tr>
<td><code>Packet &amp; operator&gt;&gt;(Int8 &amp;data)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Packet &amp; operator&gt;&gt;(Uint8 &amp;data)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Packet & operator>>(Int16 &data)
Packet & operator>>(Uint16 &data)
Packet & operator>>(Int32 &data)
Packet & operator>>(Uint32 &data)
Packet & operator>>(Int64 &data)
Packet & operator>>(Uint64 &data)
Packet & operator>>(float &data)
Packet & operator>>(double &data)
Packet & operator>>(char *data)
Packet & operator>>(std::string &data)
Packet & operator>>(wchar_t *data)
Packet & operator>>(std::wstring &data)
Packet & operator>>(String &data)

Packet & operator<<(bool data)
Overloads of operator << to write data into the packet. More
<table>
<thead>
<tr>
<th>Packet &amp; operator&lt;&lt;</th>
<th>(Int32 data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(Uint32 data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(Int64 data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(Uint64 data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(float data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(double data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(const char *data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(const std::string &amp;data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(const wchar_t *data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(const std::wstring &amp;data)</td>
</tr>
<tr>
<td>Packet &amp; operator&lt;&lt;</td>
<td>(const String &amp;data)</td>
</tr>
</tbody>
</table>
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual const void * onSend (std::size_t &amp;size)</td>
<td>Called before the packet is sent over the network. More...</td>
</tr>
<tr>
<td>virtual void onReceive (const void *data, std::size_t size)</td>
<td>Called after the packet is received over the network.</td>
</tr>
</tbody>
</table>
### Friends

<table>
<thead>
<tr>
<th>class</th>
<th>Socket Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>TcpSocket</td>
</tr>
<tr>
<td>class</td>
<td>UdpSocket</td>
</tr>
</tbody>
</table>
Detailed Description

Utility class to build blocks of data to transfer over the network.

Packets provide a safe and easy way to serialize data, in order to send (sf::TcpSocket, sf::UdpSocket).

Packets solve 2 fundamental problems that arise when transferring data over the network:

- data is interpreted correctly according to the endianness
- the bounds of the packet are preserved (one send == one receive)

The sf::Packet class provides both input and output modes. It is designed to follow the behavior of standard C++ streams, using operators >> and << to extract and insert data.

It is recommended to use only fixed-size types (like sf::Int32, etc.), to avoid differences between the sender and the receiver. Indeed, the native C++ types may have different sizes on different platforms and your data may be corrupted if that happens.

Usage example:

```cpp
sf::Uint32 x = 24;
std::string s = "hello";
double d = 5.89;

// Group the variables to send into a packet
sf::Packet packet;
packet << x << s << d;

// Send it over the network (socket is a valid sf::TcpSocket)
socket.send(packet);

// Receive the packet at the other end
sf::Packet packet;
socket.receive(packet);
```
```cpp
// Extract the variables contained in the packet
sf::Uint32 x;
std::string s;
double d;
if (packet >> x >> s >> d)
{
    // Data extracted successfully...
}
```

Packets have built-in operator >> and << overloads for standard types:

- bool
- fixed-size integer types (sf::Int8/16/32, sf::Uint8/16/32)
- floating point numbers (float, double)
- string types (char*, wchar_t*, std::string, std::wstring, sf::String)

Like standard streams, it is also possible to define your own overloads to handle your custom types.

```cpp
struct MyStruct
{
    float number;
    sf::Int8 integer;
    std::string str;
};

sf::Packet& operator <<(sf::Packet& packet, const MyStruct& m)
{
    return packet << m.number << m.integer << m.str;
}

sf::Packet& operator >>=(sf::Packet& packet, MyStruct& m)
{
    return packet >> m.number >> m.integer >> m.str;
}
```

Packets also provide an extra feature that allows to apply custom transformations to the data before it is sent, and after it is received. This is typically used to handle automatic compression or encryption of the data. This is achieved by inheriting from sf::Packet, and overriding the onSend and onReceive functions.

Here is an example:
class ZipPacket : public sf::Packet
{
    virtual const void* onSend(std::size_t& size)
    {
        const void* srcData = getData();
        std::size_t srcSize = getDataSize();
        return MySuperZipFunction(srcData, srcSize, &size);
    }
    virtual void onReceive(const void* data, std::size_t size)
    {
        std::size_t dstSize;
        const void* dstData = MySuperUnzipFunction(data, size, &dstSize);
        append(dstData, dstSize);
    }
};

// Use like regular packets:
ZipPacket packet;
packet << x << s << d;
...

See also
    sf::TcpSocket, sf::UdpSocket

Definition at line 47 of file Packet.hpp.
# Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sf::Packet::Packet()</code></td>
<td>Default constructor. Creates an empty packet.</td>
</tr>
<tr>
<td><code>virtual sf::Packet::~Packet()</code></td>
<td>Virtual destructor.</td>
</tr>
</tbody>
</table>
Member Function Documentation

```cpp
template<typename T>
void sf::Packet::append ( const void * data, std::size_t sizeInBytes )
```

Append data to the end of the packet.

**Parameters**

- `data` Pointer to the sequence of bytes to append
- `sizeInBytes` Number of bytes to append

**See also**

- clear

```cpp
template<typename T>
void sf::Packet::clear ( )
```

Clear the packet.

After calling Clear, the packet is empty.

**See also**

- append

```cpp
bool sf::Packet::endOfPacket ( ) const
```
Tell if the reading position has reached the end of the packet.

This function is useful to know if there is some data left to be read, without actually reading it.

**Returns**

- True if all data was read, false otherwise

**See also**

- operator bool

---

### `const void* sf::Packet::getData ( ) const`

Get a pointer to the data contained in the packet.

**Warning:** the returned pointer may become invalid after you append data to the packet, therefore it should never be stored. The return pointer is NULL if the packet is empty.

**Returns**

- Pointer to the data

**See also**

- `getDataSize`

---

### `std::size_t sf::Packet::getDataSize ( ) const`

Get the size of the data contained in the packet.

This function returns the number of bytes pointed to by what `getData` returns.

**Returns**
virtual void sf::Packet::onReceive (const void * data,
          std::size_t size)

Called after the packet is received over the network.

This function can be defined by derived classes to transform the data used for decompression, decryption, etc. The function receives a pointer to the received data, and it must fill the packet with the transformed bytes. The default implementation fills the packet directly without transforming the data.

**Parameters**
- **data** Pointer to the received bytes
- **size** Number of bytes

See also
- onSend

virtual const void* sf::Packet::onSend (std::size_t & size)

Called before the packet is sent over the network.

This function can be defined by derived classes to transform the data before it is sent. This can be done for compression, encryption, etc. The function must return a pointer to the modified data, as well as the number of bytes. The default implementation provides the packet's data without transforming it.
Parameters

`size` Variable to fill with the size of data to send

Returns

Pointer to the array of bytes to send

See also

onReceive

---

**sf::Packet::operator BoolType ( ) const**

Test the validity of the packet, for reading.

This operator allows to test the packet as a boolean variable, to check if a reading operation was successful.

A packet will be in an invalid state if it has no more data to read.

This behavior is the same as standard C++ streams.

Usage example:

```cpp
float x;
packet >> x;
if (packet)
{
    // ok, x was extracted successfully
}
// -- or --
float x;
if (packet >> x)
{
    // ok, x was extracted successfully
}
```

Don't focus on the return type, it's equivalent to bool but it disallows unwanted implicit conversions.
integer or pointer types.

**Returns**

True if last data extraction from packet was successful

**See also**

endOfPacket

---

### `Packet& sf::Packet::operator<<( bool data )`

Overloads of operator `<<` to write data into the packet.

### `Packet& sf::Packet::operator>>( bool & data )`

Overloads of operator `>>` to read data from the packet.

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

- Packet.hpp

---

SFML is licensed under the terms and conditions of the [zlib/png](https://zlib.net) license

Copyright © Laurent Gomila. :: Documentation generated by [doxygen](https://www.doxygen.org/)
sf::Socket Class Reference

Network module

Base class for all the socket types. More...

#include <Socket.hpp>

Inheritance diagram for sf::Socket:
### Public Types

<table>
<thead>
<tr>
<th>enum</th>
<th>Status { Done, NotReady, Partial, Disconnected, Error }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Status codes that may be returned by socket functions.</td>
</tr>
<tr>
<td>enum</td>
<td>{ AnyPort = 0 }</td>
</tr>
<tr>
<td></td>
<td>Some special values used by sockets.</td>
</tr>
</tbody>
</table>
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~Socket ()</td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>void setBlocking (bool blocking)</td>
<td>Set the blocking state of the socket. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool isBlocking () const</td>
<td>Tell whether the socket is in blocking or non-blocking mode. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
### Protected Types

<table>
<thead>
<tr>
<th>enum</th>
<th>Type { Tcp, Udp }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Types of protocols that the socket can use.</td>
</tr>
</tbody>
</table>
# Protected Member Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Socket</strong> (Type type)</td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>SocketHandle</td>
<td>getHandle () const&lt;br&gt;Return the internal handle of the socket. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>create ()&lt;br&gt;Create the internal representation of the socket. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>create (SocketHandle handle)&lt;br&gt;Create the internal representation of the socket from a socket handle. <a href="#">More...</a></td>
</tr>
<tr>
<td>void</td>
<td>close ()&lt;br&gt;Close the socket gracefully. <a href="#">More...</a></td>
</tr>
<tr>
<td>class</td>
<td>SocketSelector</td>
</tr>
</tbody>
</table>
Detailed Description

Base class for all the socket types.

This class mainly defines internal stuff to be used by derived classes.

The only public features that it defines, and which is therefore common is blocking state. All sockets can be set as blocking or non-blocking.

In blocking mode, socket functions will hang until the operation completes (well, in fact the current thread if you use multiple ones) will operation to complete.

In non-blocking mode, all the socket functions will return immediately. If the requested operation, the function simply returns the proper status code.

The default mode, which is blocking, is the one that is generally used with selectors. The non-blocking mode is rather used in real-time applications that poll the socket often enough, and cannot afford blocking this loop.

See also

sf::TcpListener, sf::TcpSocket, sf::UdpSocket

Definition at line 45 of file Socket.hpp.
### Member Enumeration Documentation

**anonymous enum**

Some special values used by sockets.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnyPort</td>
<td>Special value that tells the system to pick any available port.</td>
</tr>
</tbody>
</table>

Definition at line 66 of file `Socket.hpp`.

**enum sf::Socket::Status**

Status codes that may be returned by socket functions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>The socket has sent / received the data.</td>
</tr>
<tr>
<td>NotReady</td>
<td>The socket is not ready to send / receive data yet.</td>
</tr>
<tr>
<td>Partial</td>
<td>The socket sent a part of the data.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>The TCP socket has been disconnected.</td>
</tr>
</tbody>
</table>
An unexpected error happened.

Definition at line 53 of file `Socket.hpp`.

**enum sf::Socket::Type**

Types of protocols that the socket can use.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcp</td>
<td>TCP protocol.</td>
</tr>
<tr>
<td>Udp</td>
<td>UDP protocol.</td>
</tr>
</tbody>
</table>

Definition at line 114 of file `Socket.hpp`.
### Constructor & Destructor Documentation

**virtual sf::Socket::~Socket ()**

Destructor.

**sf::Socket::Socket ( Type type )**

Default constructor.

This constructor can only be accessed by derived classes.

**Parameters**

- **type** Type of the socket (TCP or UDP)
Member Function Documentation

**void sf::Socket::close ( )**

Close the socket gracefully.

This function can only be accessed by derived classes.

**void sf::Socket::create ( )**

Create the internal representation of the socket.

This function can only be accessed by derived classes.

**void sf::Socket::create ( SocketHandle handle )**

Create the internal representation of the socket from a socket handle.

This function can only be accessed by derived classes.

**Parameters**

- **handle** OS-specific handle of the socket to wrap

**SocketHandle sf::Socket::getHandle ( ) const**
Return the internal handle of the socket.

The returned handle may be invalid if the socket was not created yet (or already destroyed). This function can only be accessed by derived classes.

**Returns**

The internal (OS-specific) handle of the socket

---

**bool sf::Socket::isBlocking ( ) const**

Tell whether the socket is in blocking or non-blocking mode.

**Returns**

True if the socket is blocking, false otherwise

**See also**

setBlocking

---

**void sf::Socket::setBlocking ( bool blocking )**

Set the blocking state of the socket.

In blocking mode, calls will not return until they have completed their task. In blocking mode won't return until some data was actually received. Always return immediately, using the return code to signal whether there was data available or not. By default, all sockets are blocking.

**Parameters**

- **blocking** True to set the socket as blocking, false for non-blocking
See also

isBlocking

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

- Socket.hpp
Multiplexer that allows to read from multiple sockets. More...

#include <SocketSelector.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SocketSelector ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>SocketSelector (const SocketSelector &amp;copy)</strong></td>
<td>Copy constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>~SocketSelector ()</strong></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void add (Socket &amp;socket)</strong></td>
<td>Add a new socket to the selector. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void remove (Socket &amp;socket)</strong></td>
<td>Remove a socket from the selector. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>void clear ()</strong></td>
<td>Remove all the sockets stored in the selector. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool wait (Time timeout=Time::Zero)</strong></td>
<td>Wait until one or more sockets are ready to receive. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool isReady (Socket &amp;socket) const</strong></td>
<td>Test a socket to know if it is ready to receive data. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>SocketSelector &amp; operator= (const SocketSelector &amp;right)</strong></td>
<td>Overload of assignment operator. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Multiplexer that allows to read from multiple sockets.

Socket selectors provide a way to wait until some data is available on a set of sockets, instead of just one. This is convenient when you have multiple sockets that may possibly receive data, but you don't know which one will be ready first. In particular, it avoids to use a thread for each socket; a single thread can handle all the sockets.

All types of sockets can be used in a selector:

- sf::TcpListener
- sf::TcpSocket
- sf::UdpSocket

A selector doesn't store its own copies of the sockets (socket classes are not copyable anyway), it simply keeps a reference to the original sockets that you pass to the "add" function. Therefore, you can't use the selector as a socket container, you must store them outside and make sure that they are alive as long as they are used in the selector.

Using a selector is simple:

- populate the selector with all the sockets that you want to observe
- make it wait until there is data available on any of the sockets
- test each socket to find out which ones are ready

Usage example:

```cpp
// Create a socket to listen to new connections
sf::TcpListener listener;
listener.listen(55001);
```
// Create a list to store the future clients
std::list<sf::TcpSocket*> clients;

// Create a selector
sf::SocketSelector selector;

// Add the listener to the selector
selector.add(listener);

// Endless loop that waits for new connections
while (running)
{
    // Make the selector wait for data on any socket
    if (selector.wait())
    {
        // Test the listener
        if (selector.isReady(listener))
        {
            // The listener is ready: there is a pending connection
            sf::TcpSocket* client = new sf::TcpSocket;
            if (listener.accept(*client) == sf::Socket::Done)
            {
                // Add the new client to the clients list
                clients.push_back(client);
                // Add the new client to the selector so that we will
                // be notified when he sends something
                selector.add(*client);
            }
            else
            {
                // Error, we won't get a new connection, delete the socket
                delete client;
            }
        }
        else
        {
            // The listener socket is not ready, test all other sockets (the clients)
            for (std::list<sf::TcpSocket*>::iterator it = clients.begin(); it != clients.end(); ++it)
            {
                sf::TcpSocket& client = **it;
                if (selector.isReady(client))
                {
                    // The client has sent some data, we can receive it
                    sf::Packet packet;
                    if (client.receive(packet) == sf::Socket::Done)
                    {
                        ...  
                    }
                }
            }
        }
    }
}
See also

sf::Socket

Definition at line 43 of file SocketSelector.hpp.
## Constructor & Destructor Documentation

### sf::SocketSelector::SocketSelector ( )

Default constructor.

### sf::SocketSelector::SocketSelector ( const SocketSelector & copy )

Copy constructor.

**Parameters**

- **copy** Instance to copy

### sf::SocketSelector::~SocketSelector ( )

Destructor.
Member Function Documentation

**void sf::SocketSelector::add ( Socket & socket )**

Add a new socket to the selector.

This function keeps a weak reference to the socket, so you have to make sure it is not destroyed while it is stored in the selector. This function does nothing if the socket is not valid.

**Parameters**
- socket Reference to the socket to add

**See also**
- remove, clear

**void sf::SocketSelector::clear ( )**

Remove all the sockets stored in the selector.

This function doesn't destroy any instance, it simply removes all the references to external sockets.

**See also**
- add, remove

**bool sf::SocketSelector::isReady ( Socket & socket ) const**
Test a socket to know if it is ready to receive data.

This function must be used after a call to Wait, to know which sockets are ready: if a socket is ready, a call to receive will never block because we know that there is data available to read.

Note that if this function returns true for a TcpListener, this means that it is ready to accept a new connection.

**Parameters**

socket Socket to test

**Returns**

True if the socket is ready to read, false otherwise

**See also**

isReady

---

**sf::SocketSelector::operator=**

Overload of assignment operator.

**Parameters**

right Instance to assign

**Returns**

Reference to self

---

**sf::SocketSelector::remove ( Socket & socket )**

Remove a socket from the selector.

This function doesn't destroy the socket, it simply removes the reference...
Parameters

socket  Reference to the socket to remove

See also

add, clear

bool sf::SocketSelector::wait ( Time timeout = Time::Zero )

Wait until one or more sockets are ready to receive.

This function returns as soon as at least one socket has some data available. To know which sockets are ready, use the isReady function. If you use a timeout and the timeout is over, the function returns false.

Parameters

timeout  Maximum time to wait, (use Time::Zero for infinity)

Returns

True if there are sockets ready, false otherwise

See also

isReady

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

- SocketSelector.hpp

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila  ::  Documentation generated by doxygen
sf::TcpListener Class Reference

Network module

Socket that listens to new TCP connections. More...

#include <TcpListener.hpp>

Inheritance diagram for sf::TcpListener:
Public Types

| enum | Status { |
|      |   Done, NotReady, Partial, Disconnected, |
|      |   Error |
|      | } |
|      | Status codes that may be returned by socket functions. More... |

| enum | { AnyPort = 0 } |
|      | Some special values used by sockets. More... |
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TcpListener()</strong>&lt;br&gt;Default constructor. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><strong>unsigned short getLocalPort() const</strong>&lt;br&gt;Get the port to which the socket is bound locally. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><strong>Status listen(unsigned short port, const IpAddress &amp;address=IpAddress::Any)</strong>&lt;br&gt;Start listening for connections. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><strong>void close()</strong>&lt;br&gt;Stop listening and close the socket. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><strong>Status accept(TcpSocket &amp;socket)</strong>&lt;br&gt;Accept a new connection. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><strong>void setBlocking(bool blocking)</strong>&lt;br&gt;Set the blocking state of the socket. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td><strong>bool isBlocking() const</strong>&lt;br&gt;Tell whether the socket is in blocking or non-blocking mode.</td>
<td></td>
</tr>
</tbody>
</table>
Protected Types

```c
enum Type { Tcp, Udp }
```

Types of protocols that the socket can use. More...
### Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketHandle</td>
<td><strong>getHandle () const</strong>&lt;br&gt;Return the internal handle of the socket. <strong>More...</strong></td>
</tr>
<tr>
<td>void create ()</td>
<td>Create the internal representation of the socket. <strong>More...</strong></td>
</tr>
<tr>
<td>void create (SocketHandle handle)</td>
<td>Create the internal representation of the socket from a so</td>
</tr>
</tbody>
</table>
Detailed Description

Socket that listens to new TCP connections.

A listener socket is a special type of socket that listens to a given port and waits for connections on that port.

This is all it can do.

When a new connection is received, you must call accept and the listener socket will return a new instance of sf::TcpSocket that is properly initialized and can be used to communicate.

Listener sockets are specific to the TCP protocol, UDP sockets are connectionless and can communicate directly. As a consequence, a listener socket will always return sf::TcpSocket instances.

A listener is automatically closed on destruction, like all other types of sockets. If you want to stop listening before the socket is destroyed, you can call its close() function.

Usage example:

```cpp
// Create a listener socket and make it wait for new
// connections on port 55001
sf::TcpListener listener;
listener.listen(55001);

// Endless loop that waits for new connections
while (running)
{
    sf::TcpSocket client;
    if (listener.accept(client) == sf::Socket::Done)
    {
        // A new client just connected!
        std::cout << "New connection received from " << client.getRemoteAddress()
        doSomethingWith(client);
    }
}
```
See also

sf::TcpSocket, sf::Socket

Definition at line 44 of file TcpListener.hpp.
Member Enumeration Documentation

**anonymous enum**

Some special values used by sockets.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnyPort</td>
<td>Special value that tells the system to pick any available port.</td>
</tr>
</tbody>
</table>

Definition at line 66 of file `Socket.hpp`.

**enum sf::Socket::Status**

Status codes that may be returned by socket functions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>The socket has sent / received the data.</td>
</tr>
<tr>
<td>NotReady</td>
<td>The socket is not ready to send / receive data yet.</td>
</tr>
<tr>
<td>Partial</td>
<td>The socket sent a part of the data.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>The TCP socket has been disconnected.</td>
</tr>
</tbody>
</table>
An unexpected error happened.

Definition at line 53 of file Socket.hpp.

**enum sf::Socket::Type**

Types of protocols that the socket can use.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcp</td>
<td>TCP protocol.</td>
</tr>
<tr>
<td>Udp</td>
<td>UDP protocol.</td>
</tr>
</tbody>
</table>

Definition at line 114 of file Socket.hpp.
sf::TcpListener::TcpListener ( )

Default constructor.
### Member Function Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status sf::TcpListener::accept ( TcpSocket &amp; socket )</strong></td>
<td>Accept a new connection. If the socket is in blocking mode, this function will not return until a connection is actually received.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>socket Socket</td>
<td>that will hold the new connection</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
<td></td>
</tr>
<tr>
<td>Status code</td>
<td></td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td></td>
</tr>
<tr>
<td>listen</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void sf::TcpListener::close ( )</strong></td>
<td>Stop listening and close the socket. This function gracefully stops the listener. If the socket is not listening, this function has no effect.</td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td></td>
</tr>
<tr>
<td>listen</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void sf::Socket::create ( )</strong></td>
<td></td>
</tr>
</tbody>
</table>
Create the internal representation of the socket.
This function can only be accessed by derived classes.

```cpp
void sf::Socket::create (SocketHandle handle)
```

Create the internal representation of the socket from a socket handle.
This function can only be accessed by derived classes.

**Parameters**

- `handle` OS-specific handle of the socket to wrap

```cpp
SocketHandle sf::Socket::getHandle ( ) const
```

Return the internal handle of the socket.
The returned handle may be invalid if the socket was not created
function can only be accessed by derived classes.

**Returns**

The internal (OS-specific) handle of the socket

```cpp
unsigned short sf::TcpListener::getLocalPort ( ) const
```

Get the port to which the socket is bound locally.
If the socket is not listening to a port, this function returns 0.
**Returns**
Port to which the socket is bound

**See also**
listen

**bool sf::Socket::isBlocking() const**

Tell whether the socket is in blocking or non-blocking mode.

**Returns**
True if the socket is blocking, false otherwise

**See also**
setBlocking

**Status sf::TcpListener::listen(unsigned short port, const IpAddress &address = IpAddress::Any)**

Start listening for connections.

This function makes the socket listen to the specified port, waiting for new connections. If the socket was previously listening to another port, it will be stopped first and bound.

**Parameters**

- `port` Port to listen for new connections
- `address` Address of the interface to listen on

**Returns**
Status code
See also
accept, close

`void sf::Socket::setBlocking ( bool blocking )`

Set the blocking state of the socket.

In blocking mode, calls will not return until they have completed their task. In blocking mode, calls won't return until some data was actually received. Always return immediately, using the return code to signal whether there was data available or not. By default, all sockets are blocking.

Parameters

- **blocking** True to set the socket as blocking, false for non-blocking

See also

- isBlocking

The documentation for this class was generated from the following file:
- TcpListener.hpp

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila  ::  Documentation generated by doxygen
sf::TcpSocket Class Reference

Network module

Specialized socket using the TCP protocol. More...

#include <TcpSocket.hpp>

Inheritance diagram for sf::TcpSocket:
## Public Types

| enum | Status { |
|      | Done, NotReady, Partial, Disconnected, Error |
|      | } |
|      | Status codes that may be returned by socket functions. [More...](#) |

| enum | { AnyPort = 0 } |
|      | Some special values used by sockets. [More...](#) |
## Public Member Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TcpSocket ()</strong></td>
<td>Default constructor.</td>
</tr>
<tr>
<td><strong>getLocalPort () const</strong></td>
<td>Get the port to which the socket is bound locally.</td>
</tr>
<tr>
<td><strong>getRemoteAddress () const</strong></td>
<td>Get the address of the connected peer.</td>
</tr>
<tr>
<td><strong>getRemotePort () const</strong></td>
<td>Get the port of the connected peer to which the socket is connected.</td>
</tr>
<tr>
<td><strong>connect (const IpAddress &amp;remoteAddress, unsigned short remotePort, timeout=Time::Zero)</strong></td>
<td>Connect the socket to a remote peer.</td>
</tr>
<tr>
<td><strong>disconnect ()</strong></td>
<td>Disconnect the socket from its remote peer.</td>
</tr>
<tr>
<td>*<em>send (const void <em>data, std::size_t size)</em></em></td>
<td>Send raw data to the remote peer.</td>
</tr>
<tr>
<td>*<em>send (const void <em>data, std::size_t size, std::size_t &amp;sent)</em></em></td>
<td>Send raw data to the remote peer.</td>
</tr>
<tr>
<td>*<em>receive (void <em>data, std::size_t size, std::size_t &amp;received)</em></em></td>
<td>Receive raw data from the remote peer.</td>
</tr>
<tr>
<td><strong>send (Packet &amp;packet)</strong></td>
<td>Send a formatted packet of data to the remote peer.</td>
</tr>
</tbody>
</table>
### Status

**receive (Packet &packet)**

Receive a formatted packet of data from the remote peer.

### Void

**void setBlocking (bool blocking)**

Set the blocking state of the socket. [More...](#)

### Bool

**bool isBlocking () const**

Tell whether the socket is in blocking or non-blocking mode.
## Protected Types

```c
enum Type { Tcp, Udp }
```

Types of protocols that the socket can use. [More...](#)
## Protected Member Functions

<table>
<thead>
<tr>
<th>SocketHandle</th>
<th>getHandle () const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return the internal handle of the socket. <a href="#">More...</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>create ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create the internal representation of the socket. <a href="#">More...</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>create (SocketHandle handle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create the internal representation of the socket from a so</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>close ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close the socket gracefully. <a href="#">More...</a></td>
<td></td>
</tr>
</tbody>
</table>

[More...](#)
class **TcpListener**
Detailed Description

Specialized socket using the TCP protocol.

TCP is a connected protocol, which means that a TCP socket can only connected to.

It can’t send or receive anything if it is not connected.

The TCP protocol is reliable but adds a slight overhead. It ensures that your data will always be received in order and without errors (no data corrupted, lost or duplicated).

When a socket is connected to a remote host, you can retrieve information about this host with the getRemoteAddress and getRemotePort functions. You can also get the local port to which the socket is bound (which is automatically chosen when the socket is connected), with the getLocalPort function.

Sending and receiving data can use either the low-level or the high-level interface. The low-level functions process a raw sequence of bytes, and cannot ensure that one call to Send will exactly match one call to Receive at the other end of the socket.

The high-level interface uses packets (see sf::Packet), which are easier to use and provide more safety regarding the data that is exchanged. You can look at the sf::Packet class to get more details about how they work.

The socket is automatically disconnected when it is destroyed, but if you want to explicitly close the connection while the socket instance is still alive, you can call disconnect.

Usage example:

```cpp
// ----- The client -----  

// Create a socket and connect it to 192.168.1.50 on port 55001 
sf::TcpSocket socket;
```
socket.connect("192.168.1.50", 55001);

// Send a message to the connected host
std::string message = "Hi, I am a client";
socket.send(message.c_str(), message.size() + 1);

// Receive an answer from the server
char buffer[1024];
std::size_t received = 0;
socket.receive(buffer, sizeof(buffer), received);
std::cout << "The server said: " << buffer << std::endl;

// ----- The server ----- 

// Create a listener to wait for incoming connections on port 55001
sf::TcpListener listener;
listener.listen(55001);

// Wait for a connection
sf::TcpSocket socket;
listener.accept(socket);
std::cout << "New client connected: " << socket.getRemoteAddress() << std::endl;

// Receive a message from the client
char buffer[1024];
std::size_t received = 0;
socket.receive(buffer, sizeof(buffer), received);
std::cout << "The client said: " << buffer << std::endl;

// Send an answer
std::string message = "Welcome, client";
socket.send(message.c_str(), message.size() + 1);

See also
sf::Socket, sf::UdpSocket, sf::Packet

Definition at line 46 of file TcpSocket.hpp.
### Member Enumeration Documentation

#### anonymous enum

Some special values used by sockets.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnyPort</td>
<td>Special value that tells the system to pick any available port.</td>
</tr>
</tbody>
</table>

Definition at line 66 of file `Socket.hpp`.

#### enum sf::Socket::Status

Status codes that may be returned by socket functions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>The socket has sent / received the data.</td>
</tr>
<tr>
<td>NotReady</td>
<td>The socket is not ready to send / receive data yet.</td>
</tr>
<tr>
<td>Partial</td>
<td>The socket sent a part of the data.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>The TCP socket has been disconnected.</td>
</tr>
</tbody>
</table>
An unexpected error happened.

Definition at line 53 of file `Socket.hpp`.

```cpp
enum sf::Socket::Type
```

Types of protocols that the socket can use.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcp</td>
<td>TCP protocol.</td>
</tr>
<tr>
<td>Udp</td>
<td>UDP protocol.</td>
</tr>
</tbody>
</table>

Definition at line 114 of file `Socket.hpp`. 
Constructor & Destructor Documentation

sf::TcpSocket::TcpSocket ( )

Default constructor.
Member Function Documentation

### void sf::Socket::close()

Close the socket gracefully.

This function can only be accessed by derived classes.

### Status sf::TcpSocket::connect (const IpAddress & remoteAddress, unsigned short remotePort, Time timeout = Time::Zero)

Connect the socket to a remote peer.

In blocking mode, this function may take a while, especially if the remote peer is not reachable. The last parameter allows you to stop trying to connect after a given timeout. If the socket was previously connected, it is first disconnected.

**Parameters**
- **remoteAddress**: Address of the remote peer
- **remotePort**: Port of the remote peer
- **timeout**: Optional maximum time to wait

**Returns**
- Status code

**See also**
void sf::Socket::create()

Create the internal representation of the socket.

This function can only be accessed by derived classes.

void sf::Socket::create(SocketHandle handle)

Create the internal representation of the socket from a socket handle.

This function can only be accessed by derived classes.

Parameters

handle OS-specific handle of the socket to wrap

void sf::TcpSocket::disconnect()

Disconnect the socket from its remote peer.

This function gracefully closes the connection. If the socket is not connected, it has no effect.

See also

connect

SocketHandle sf::Socket::getHandle() const
Return the internal handle of the socket.

The returned handle may be invalid if the socket was not created yet or already destroyed. This function can only be accessed by derived classes.

**Returns**
- The internal (OS-specific) handle of the socket

---

**unsigned short sf::TcpSocket::getLocalPort ( ) const**

Get the port to which the socket is bound locally.

If the socket is not connected, this function returns 0.

**Returns**
- Port to which the socket is bound

**See also**
- connect, getRemotePort

---

**IpAddress sf::TcpSocket::getRemoteAddress ( ) const**

Get the address of the connected peer.

If the socket is not connected, this function returns `sf::IpAddress::None`.

**Returns**
- Address of the remote peer

**See also**
getRemotePort

**unsigned short sf::TcpSocket::getRemotePort ( ) const**

Get the port of the connected peer to which the socket is connected.

If the socket is not connected, this function returns 0.

**Returns**
Remote port to which the socket is connected

**See also**
getRemoteAddress

bool sf::Socket::isBlocking ( ) const

Tell whether the socket is in blocking or non-blocking mode.

**Returns**
True if the socket is blocking, false otherwise

**See also**
setBlocking

**Status sf::TcpSocket::receive ( void * data, std::size_t size, std::size_t & received )**
Receive raw data from the remote peer.

In blocking mode, this function will wait until some bytes are actually received. This function will fail if the socket is not connected.

### Parameters
- **data**: Pointer to the array to fill with the received bytes
- **size**: Maximum number of bytes that can be received
- **received**: This variable is filled with the actual number of bytes received

### Returns
- Status code

### See also
- send

---

**Status sf::TcpSocket::receive ( Packet & packet )**

Receive a formatted packet of data from the remote peer.

In blocking mode, this function will wait until the whole packet has been received. This function will fail if the socket is not connected.

### Parameters
- **packet**: Packet to fill with the received data

### Returns
- Status code

### See also
- send
**Status** sf::TcpSocket::send (const void * data, std::size_t size)

Send raw data to the remote peer.

To be able to handle partial sends over non-blocking sockets, use the std::size_t overload instead. This function will fail if the socket is not connected.

**Parameters**
- **data** Pointer to the sequence of bytes to send
- **size** Number of bytes to send

**Returns**
- Status code

**See also**
- receive

**Status** sf::TcpSocket::send (const void * data, std::size_t size, std::size_t &sent)

Send raw data to the remote peer.

This function will fail if the socket is not connected.

**Parameters**
- **data** Pointer to the sequence of bytes to send
- **size** Number of bytes to send
The number of bytes sent will be written here

**Returns**

Status code

**See also**

receive

---

**Status** sf::TcpSocket::send ( Packet & packet )

Send a formatted packet of data to the remote peer.

In non-blocking mode, if this function returns `sf::Socket::Partial`, you unmodified packet before sending anything else in order to guarantee peer uncorrupted. This function will fail if the socket is not connected.

**Parameters**

packet Packet to send

**Returns**

Status code

**See also**

receive

---

void sf::Socket::setBlocking ( bool blocking )

Set the blocking state of the socket.

In blocking mode, calls will not return until they have completed their task. In blocking mode won't return until some data was actually received.
always return immediately, using the return code to signal whether there was data available or not. By default, all sockets are blocking.

**Parameters**

- **blocking** True to set the socket as blocking, false for non-blocking

**See also**

- isBlocking

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

- TcpSocket.hpp
sf::UdpSocket Class Reference
Network module

Specialized socket using the UDP protocol. More...

#include <UdpSocket.hpp>

Inheritance diagram for sf::UdpSocket:
### Public Types

**enum**

| MaxDatagramSize = 65507 |

**enum**

<table>
<thead>
<tr>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done, NotReady, Partial, Disconnected, Error</td>
</tr>
</tbody>
</table>

Status codes that may be returned by socket functions. More...

**enum**

| AnyPort = 0 |

Some special values used by sockets. More...
## Public Member Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>UdpSocket ()</td>
<td>Default constructor.</td>
<td></td>
</tr>
<tr>
<td>unsigned short</td>
<td>getLocalPort () const</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>bind (unsigned short port, const IpAddress &amp;address=IpAddress::Any)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>unbind ()</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>send (const void *data, std::size_t size, const IpAddress &amp;remotePort)</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>receive (void *data, std::size_t size, std::size_t &amp;received, unsigned short &amp;remotePort)</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>send (Packet &amp;packet, const IpAddress &amp;remoteAddress)</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>receive (Packet &amp;packet, IpAddress &amp;remoteAddress, unsigned short &amp;remotePort)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>setBlocking (bool blocking)</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>isBlocking () const</td>
<td></td>
</tr>
</tbody>
</table>
Protected Types

```cpp
enum Type { Tcp, Udp }
Types of protocols that the socket can use. More...
```
## Protected Member Functions

<table>
<thead>
<tr>
<th>SocketHandle</th>
<th>getHandle () const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return the internal handle of the socket.</td>
<td>More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>create ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create the internal representation of the socket.</td>
<td>More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>create (SocketHandle handle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create the internal representation of the socket from a so</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>close ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close the socket gracefully.</td>
<td>More...</td>
</tr>
</tbody>
</table>
Detailed Description

Specialized socket using the UDP protocol.

A UDP socket is a connectionless socket.

Instead of connecting once to a remote host, like TCP sockets, it can ser any time.

It is a datagram protocol: bounded blocks of data (datagrams) are transfer continuous stream of data (TCP). Therefore, one call to send will always datagram is not lost), with the same data that was sent.

The UDP protocol is lightweight but unreliable. Unreliable means that data or arrive reordered. However, if a datagram arrives, its data is guaranteed

UDP is generally used for real-time communication (audio or video stream speed is crucial and lost data doesn't matter much.

Sending and receiving data can use either the low-level or the high-level process a raw sequence of bytes, whereas the high-level interface uses are easier to use and provide more safety regarding the data that is sf::Packet class to get more details about how they work.

It is important to note that UdpSocket is unable to send datagrams bigger case, it returns an error and doesn't send anything. This applies to both even packets are unable to split and recompose data, due to the unreliable mixed or duplicated datagrams may lead to a big mess when trying to recon.

If the socket is bound to a port, it is automatically unbound from it when the you can unbind the socket explicitly with the Unbind function if necessary.
make the port available for other sockets.

Usage example:

```cpp
// ----- The client -----  
// Create a socket and bind it to the port 55001
sf::UdpSocket socket;
socket.bind(55001);

// Send a message to 192.168.1.50 on port 55002
std::string message = "Hi, I am " + sf::IpAddress::getLocalAddress().toString();
socket.send(message.c_str(), message.size() + 1, "192.168.1.50", 55002);

// Receive an answer (most likely from 192.168.1.50, but could be anyone)
char buffer[1024];
std::size_t received = 0;
sf::IpAddress sender;
unsigned short port;
socket.receive(buffer, sizeof(buffer), received, sender, port);
std::cout << sender.ToString() << " said: " << buffer << std::endl;

// ----- The server -----  
// Create a socket and bind it to the port 55002
sf::UdpSocket socket;
socket.bind(55002);

// Receive a message from anyone
char buffer[1024];
std::size_t received = 0;
sf::IpAddress sender;
unsigned short port;
socket.receive(buffer, sizeof(buffer), received, sender, port);
std::cout << sender.ToString() << " said: " << buffer << std::endl;

// Send an answer
std::string message = "Welcome " + sender.toString();
socket.send(message.c_str(), message.size() + 1, sender, port);
```

See also
sf::Socket, sf::TcpSocket, sf::Packet

Definition at line 45 of file UdpSocket.hpp.
Member Enumeration Documentation

**anonymous enum**

Some special values used by sockets.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnyPort</td>
<td>Special value that tells the system to pick any available port.</td>
</tr>
</tbody>
</table>

Definition at line 66 of file `Socket.hpp`.

**anonymous enum**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxDatagramSize</td>
<td>The maximum number of bytes that can be sent in a single UDP datagram.</td>
</tr>
</tbody>
</table>

Definition at line 52 of file `UdpSocket.hpp`.

**enum sf::Socket::Status**

Status codes that may be returned by socket functions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td></td>
</tr>
</tbody>
</table>
The socket has sent / received the data.

<table>
<thead>
<tr>
<th>NotReady</th>
<th>The socket is not ready to send / receive data yet.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>The socket sent a part of the data.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>The TCP socket has been disconnected.</td>
</tr>
<tr>
<td>Error</td>
<td>An unexpected error happened.</td>
</tr>
</tbody>
</table>

Definition at line 53 of file `Socket.hpp`.

```
enum sf::Socket::Type
```

Types of protocols that the socket can use.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcp</td>
<td>TCP protocol.</td>
</tr>
<tr>
<td>Udp</td>
<td>UDP protocol.</td>
</tr>
</tbody>
</table>

Definition at line 114 of file `Socket.hpp`. 

sf::UdpSocket::UdpSocket ()

Default constructor.
## Member Function Documentation

**Status sf::UdpSocket::bind (unsigned short port, const IpAddress & address = IpAddress::Any)***

Bind the socket to a specific port.

Binding the socket to a port is necessary for being able to receive data. Use the special value `Socket::AnyPort` to tell the system to automatically pick a port, and then call `getLocalPort` to retrieve the chosen port.

**Parameters**

- **port** Port to bind the socket to
- **address** Address of the interface to bind to

**Returns**

Status code

**See also**

`unbind`, `getLocalPort`

---

**void sf::Socket::close ()***

Close the socket gracefully.

This function can only be accessed by derived classes.
void sf::Socket::create ( )

Create the internal representation of the socket.
This function can only be accessed by derived classes.

void sf::Socket::create ( SocketHandle handle )

Create the internal representation of the socket from a socket handle.
This function can only be accessed by derived classes.

Parameters

handle OS-specific handle of the socket to wrap

SocketHandle sf::Socket::getHandle ( ) const

Return the internal handle of the socket.
The returned handle may be invalid if the socket was not created and this function can only be accessed by derived classes.

Returns

The internal (OS-specific) handle of the socket

unsigned short sf::UdpSocket::getLocalPort ( ) const

Get the port to which the socket is bound locally.
If the socket is not bound to a port, this function returns 0.

**Returns**
Port to which the socket is bound

**See also**
bind

```cpp
bool sf::Socket::isBlocking ( ) const
```

Tell whether the socket is in blocking or non-blocking mode.

**Returns**
True if the socket is blocking, false otherwise

**See also**
setBlocking

```cpp
Status sf::UdpSocket::receive ( void *data,
   std::size_t size,
   std::size_t &received,
   IpAddress &remoteAddress,
   unsigned short &remotePort )
```

Receive raw data from a remote peer.

In blocking mode, this function will wait until some bytes are actually received which is large enough for the data that you intend to receive, if it is returned and all the data will be lost.
Parameters

- **data**: Pointer to the array to fill with the received bytes
- **size**: Maximum number of bytes that can be received
- **received**: This variable is filled with the actual number of bytes received
- **remoteAddress**: Address of the peer that sent the data
- **remotePort**: Port of the peer that sent the data

Returns

- Status code

See also

- send

**sf::UdpSocket::receive**

```cpp
Status sf::UdpSocket::receive ( Packet & packet,
                               IpAddress & remoteAddress,
                               unsigned short & remotePort )
```

Receive a formatted packet of data from a remote peer.

In blocking mode, this function will wait until the whole packet has been received.

Parameters

- **packet**: Packet to fill with the received data
- **remoteAddress**: Address of the peer that sent the data
- **remotePort**: Port of the peer that sent the data

Returns

- Status code

See also

- send
### sf::UdpSocket::send

**Status**

```cpp
sf::UdpSocket::send ( const void * data,
                     std::size_t size,
                     const IpAddress & remoteAddress,
                     unsigned short remotePort )
```

Send raw data to a remote peer.

Make sure that `size` is not greater than `UdpSocket::MaxDatagramSize` and no data will be sent.

#### Parameters

- **data**: Pointer to the sequence of bytes to send
- **size**: Number of bytes to send
- **remoteAddress**: Address of the receiver
- **remotePort**: Port of the receiver to send the data to

#### Returns

Status code

#### See also

receive

### sf::UdpSocket::send

```cpp
sf::UdpSocket::send ( Packet & packet,
                      const IpAddress & remoteAddress,
                      unsigned short remotePort )
```

Send a formatted packet of data to a remote peer.

Make sure that the packet size is not greater than `UdpSocket::MaxDatagramSize`
will fail and no data will be sent.

Parameters
- `packet` : Packet to send
- `remoteAddress` : Address of the receiver
- `remotePort` : Port of the receiver to send the data to

Returns
- Status code

See also
- `receive`

---

### void sf::Socket::setBlocking ( bool blocking )

Set the blocking state of the socket.

In blocking mode, calls will not return until they have completed their task in blocking mode won't return until some data was actually received. Always return immediately, using the return code to signal whether there was data available or not. By default, all sockets are blocking.

Parameters
- `blocking` : True to set the socket as blocking, false for non-blocking

See also
- `isBlocking`

---

### void sf::UdpSocket::unbind ( )

Unbind the socket from the local port to which it is bound.
The port that the socket was previously bound to is immediately made available after this function is called. This means that a subsequent call to `bind()` will be able to re-bind the port if no other process has done so in the mean time. If the socket is not bound to a port, this function has no effect.

**See also**

`bind`

The documentation for this class was generated from the following file:
- `UdpSocket.hpp`
System module

Base module of SFML, defining various utilities. More...
## Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf::Clock</td>
<td>Utility class that measures the elapsed time. <a href="#">More...</a></td>
</tr>
<tr>
<td>sf::FileInputStream</td>
<td>Implementation of input stream based on a file. <a href="#">More...</a></td>
</tr>
<tr>
<td>sf::InputStream</td>
<td>Abstract class for custom file input streams. <a href="#">More...</a></td>
</tr>
<tr>
<td>sf::Lock</td>
<td>Automatic wrapper for locking and unlocking mutexes. <a href="#">More...</a></td>
</tr>
<tr>
<td>sf::MemoryInputStream</td>
<td>Implementation of input stream based on a memory chunk. <a href="#">More</a></td>
</tr>
<tr>
<td>sf::Mutex</td>
<td>Blocks concurrent access to shared resources from multiple threads.</td>
</tr>
<tr>
<td>sf::NonCopyable</td>
<td>Utility class that makes any derived class non-copyable. <a href="#">More...</a></td>
</tr>
<tr>
<td>sf::String</td>
<td>Utility string class that automatically handles conversions between types.</td>
</tr>
<tr>
<td>sf::Thread</td>
<td>Utility class to manipulate threads. <a href="#">More...</a></td>
</tr>
<tr>
<td>sf::ThreadLocal</td>
<td>Defines variables with thread-local storage. <a href="#">More...</a></td>
</tr>
<tr>
<td>sf::ThreadLocalPtr&lt; T &gt;</td>
<td></td>
</tr>
</tbody>
</table>

[More...](#)
<table>
<thead>
<tr>
<th><strong>class</strong></th>
<th><strong>sf::Time</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Represents a time value. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>class</strong></th>
<th><strong>sf::Utf&lt; N &gt;</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utility class providing generic functions for UTF conversions. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>class</strong></th>
<th><strong>sf::Vector2&lt; T &gt;</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utility template class for manipulating 2-dimensional vectors. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>class</strong></th>
<th><strong>sf::Vector3&lt; T &gt;</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utility template class for manipulating 3-dimensional vectors. More...</td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANativeActivity * sf::getNativeActivity ()</td>
<td>Return a pointer to the Android native activity.</td>
<td></td>
</tr>
<tr>
<td>void sf::sleep (Time duration)</td>
<td>Make the current thread sleep for a given duration.</td>
<td></td>
</tr>
<tr>
<td>std::ostream &amp; sf::err ()</td>
<td>Standard stream used by SFML to output warnings and errors.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Base module of SFML, defining various utilities.

It provides vector classes, Unicode strings and conversion functions, three
Function Documentation

**sf::err()**

Standard stream used by SFML to output warnings and errors.

By default, *sf::err()* outputs to the same location as *std::cerr*, (↔ the console if there's one available.

It is a standard *std::ostream* instance, so it supports all the insertion (operator <<, manipulators, etc.).

*sf::err()* can be redirected to write to another output, independently of function provided by the *std::ostream* class.

Example:

```cpp
// Redirect to a file
std::ofstream file("sfml-log.txt");
std::streambuf* previous = sf::err().rdbuf(file.rdbuf());

// Redirect to nothing
sf::err().rdbuf(NULL);

// Restore the original output
sf::err().rdbuf(previous);
```

**Returns**

Reference to *std::ostream* representing the SFML error stream

**ANativeActivity* sf::getNativeActivity ( )**
Return a pointer to the Android native activity.

You shouldn't have to use this function, unless you want to implement details that SFML doesn't support, or to use a workaround for a known issue.

**Returns**

- Pointer to Android native activity structure

**Platform Limitation**

This is only available on Android and to use it, you'll have to specifically include SFML/System/NativeActivity.hpp in your code.

---

```cpp
void sf::sleep ( Time duration )
```

Make the current thread sleep for a given duration.

sf::sleep is the best way to block a program or one of its threads, as it does not consume any CPU power.

**Parameters**

- `duration` Time to sleep

---

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Clock Class Reference

System module

Utility class that measures the elapsed time. More...

#include <Clock.hpp>
## Public Member Functions

<table>
<thead>
<tr>
<th>Class</th>
<th>Function Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Clock ()</td>
<td>Default constructor.</td>
<td>More...</td>
</tr>
<tr>
<td>Time</td>
<td>getTimeElapsed ()</td>
<td>Get the elapsed time.</td>
<td>More...</td>
</tr>
<tr>
<td>Time</td>
<td>restart ()</td>
<td>Restart the clock.</td>
<td>More...</td>
</tr>
</tbody>
</table>
Detailed Description

Utility class that measures the elapsed time.

`sf::Clock` is a lightweight class for measuring time.

It provides the most precise time that the underlying OS can achieve (generally microseconds or nanoseconds). It also ensures monotonicity, which means that the returned time can never go backward, even if the system time is changed.

Usage example:

```cpp
sf::Clock clock;
...
Time time1 = clock.getElapsedTime();
...
Time time2 = clock.restart();
```

The `sf::Time` value returned by the clock can then be converted to a number of seconds, milliseconds or even microseconds.

See also

`sf::Time`

Definition at line 41 of file `Clock.hpp`. 
sf::Clock::Clock ( )

Default constructor.

The clock starts automatically after being constructed.
### Member Function Documentation

<table>
<thead>
<tr>
<th>Time sf::Clock::getElapsedTime ( ) const</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the elapsed time.</td>
</tr>
<tr>
<td>This function returns the time elapsed since the last call to restart() (or to restart() has not been called).</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td>Time elapsed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time sf::Clock::restart ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart the clock.</td>
</tr>
<tr>
<td>This function puts the time counter back to zero. It also returns the time started.</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td>Time elapsed</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following file:
- Clock.hpp

---

SFML is licensed under the terms and conditions of the zlib/png license.
Implementation of input stream based on a file. More...

#include <FileInputStream.hpp>

Inheritance diagram for sf::FileInputStream:
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FileInputStream()</code></td>
<td>Default constructor.</td>
</tr>
<tr>
<td><code>~FileInputStream()</code></td>
<td>Default destructor.</td>
</tr>
<tr>
<td><code>bool open(const std::string &amp;filename)</code></td>
<td>Open the stream from a file path.</td>
</tr>
<tr>
<td><code>virtual Int64 read(void *data, Int64 size)</code></td>
<td>Read data from the stream.</td>
</tr>
<tr>
<td><code>virtual Int64 seek(Int64 position)</code></td>
<td>Change the current reading position.</td>
</tr>
<tr>
<td><code>virtual Int64 tell()</code></td>
<td>Get the current reading position in the stream.</td>
</tr>
<tr>
<td><code>virtual Int64 getSize()</code></td>
<td>Return the size of the stream.</td>
</tr>
</tbody>
</table>
Detailed Description

Implementation of input stream based on a file.

This class is a specialization of InputStream that reads from a file on disk.

It wraps a file in the common InputStream interface and therefore a functions that accept such a stream, with a file on disk as the data source.

In addition to the virtual functions inherited from InputStream, FileInputStream the file to open.

SFML resource classes can usually be loaded directly from a filename, so you unless you create your own algorithms that operate on an InputStream

Usage example:

```cpp
void process(InputStream& stream);
FileStream stream;
if (stream.open("some_file.dat"))
    process(stream);
```

InputStream, MemoryInputStream

Definition at line 55 of file FileInputStream.hpp.
**Constructor & Destructor Documentation**

<table>
<thead>
<tr>
<th>sf::FileInputStream::FileInputStream ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default constructor.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual sf::FileInputStream::~FileInputStream ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default destructor.</td>
</tr>
</tbody>
</table>
# Member Function Documentation

**virtual Int64 sf::FileInputStream::getSize ( )**

Return the size of the stream.

**Returns**
- The total number of bytes available in the stream, or -1 on error.

Implements sf::InputStream.

**bool sf::FileInputStream::open ( const std::string & filename )**

Open the stream from a file path.

**Parameters**
- **filename** Name of the file to open

**Returns**
- True on success, false on error

**virtual Int64 sf::FileInputStream::read ( void * data, Int64 size )**

Read data from the stream.
After reading, the stream's reading position must be advanced by the amount of bytes read.

**Parameters**
- `data` Buffer where to copy the read data
- `size` Desired number of bytes to read

**Returns**
- The number of bytes actually read, or -1 on error

Implements `sf::InputStream`.

---

**virtual Int64 sf::FileInputStream::seek ( Int64 position )**

Change the current reading position.

**Parameters**
- `position` The position to seek to, from the beginning

**Returns**
- The position actually sought to, or -1 on error

Implements `sf::InputStream`.

---

**virtual Int64 sf::FileInputStream::tell ( )**

Get the current reading position in the stream.

**Returns**
- The current position, or -1 on error.

Implements `sf::InputStream`. 

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

- FileInputStream.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
Abstract class for custom file input streams. More...

#include <InputStream.hpp>

Inheritance diagram for sf::InputStream:
## Public Member Functions

<table>
<thead>
<tr>
<th>virtual</th>
<th>~InputStream ()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Virtual destructor. More...</td>
</tr>
</tbody>
</table>

| virtual Int64 | read (void *data, Int64 size)=0 |
|               | Read data from the stream. More... |

| virtual Int64 | seek (Int64 position)=0 |
|               | Change the current reading position. More... |

| virtual Int64 | tell ()=0 |
|               | Get the current reading position in the stream. More... |

| virtual Int64 | getSize ()=0 |
|               | Return the size of the stream. More... |
Detailed Description

Abstract class for custom file input streams.

This class allows users to define their own file input sources from which SFML resource classes like sf::Texture and sf::SoundBuffer provide loadFromFile and loadFromMemory functions, which read data from conventional sources. However, if you have data coming from a different source (over a network, embedded, encrypted, compressed, etc) you can derive your own class from sf::InputStream and load SFML resources with their loadFromStream function.

Usage example:

```cpp
// custom stream class that reads from inside a zip file
class ZipStream : public sf::InputStream
{
public:
    ZipStream(std::string archive);
    bool open(std::string filename);
    Int64 read(void* data, Int64 size);
    Int64 seek(Int64 position);
    Int64 tell();
    Int64 getSize();

private:
    ...};

// now you can load textures...
sf::Texture texture;
ZipStream stream("resources.zip");
stream.open("images/img.png");
texture.loadFromStream(stream);

// musics...
sf::Music music;
```
ZipStream stream("resources.zip");
stream.open("musics/msc.ogg");
music.openFromStream(stream);

// etc.

Definition at line 41 of file InputStream.hpp.
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>virtual sf::InputStream::~InputStream ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual destructor.</td>
</tr>
<tr>
<td>Definition at line 49 of file InputStream.hpp.</td>
</tr>
</tbody>
</table>
Member Function Documentation

virtual Int64 sf::InputStream::getSize ()

Return the size of the stream.

Returns
The total number of bytes available in the stream, or -1 on error

Implemented in sf::FileInputStream, and sf::MemoryInputStream.

virtual Int64 sf::InputStream::read ( void * data, Int64 size )

Read data from the stream.

After reading, the stream's reading position must be advanced by the amount of bytes read.

Parameters
  data Buffer where to copy the read data
  size Desired number of bytes to read

Returns
The number of bytes actually read, or -1 on error

Implemented in sf::FileInputStream, and sf::MemoryInputStream.
### virtual Int64 sf::InputStream::seek (Int64 position)

Change the current reading position.

**Parameters**

- **position** The position to seek to, from the beginning

**Returns**

- The position actually sought to, or -1 on error

Implemented in `sf::FileInputStream`, and `sf::MemoryInputStream`.

### virtual Int64 sf::InputStream::tell ( )

Get the current reading position in the stream.

**Returns**

- The current position, or -1 on error.

Implemented in `sf::FileInputStream`, and `sf::MemoryInputStream`.

The documentation for this class was generated from the following file:
- InputStream.hpp

---

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Lock Class Reference

System module

Automatic wrapper for locking and unlocking mutexes. More...

#include <Lock.hpp>

Inheritance diagram for sf::Lock:
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock (Mutex &amp;mutex)</td>
<td>Construct the lock with a target mutex. More...</td>
</tr>
<tr>
<td>~Lock ()</td>
<td>Destructor. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Automatic wrapper for locking and unlocking mutexes.

`sf::Lock` is a RAII wrapper for `sf::Mutex`.

By unlocking it in its destructor, it ensures that the mutex will always be released (most likely a function) ends. This is even more important when an exception can interrupt the execution flow of the function.

For maximum robustness, `sf::Lock` should always be used to lock/unlock a mutex.

Usage example:

```cpp
sf::Mutex mutex;
void function()
{
    sf::Lock lock(mutex); // mutex is now locked
    functionThatMayThrowAnException(); // mutex is unlocked if this function throws
    if (someCondition)
        return; // mutex is unlocked
} // mutex is unlocked
```

Because the mutex is not explicitly unlocked in the code, it may remain locked longer than needed. If the region of the code that needs to be protected by the mutex is not the entire function, it's a good practice to create a smaller, inner scope so that the lock is limited to this part of the code.

```cpp
sf::Mutex mutex;
void function()
{
    {
        sf::Lock lock(mutex);
        codeThatRequiresProtection();
    }
}
```
Having a mutex locked longer than required is a bad practice which can forget that when a mutex is locked, other threads may be waiting doing no

**See also**

sf::Mutex

Definition at line 43 of file Lock.hpp.
**sf::Lock::Lock ( Mutex & mutex )**

Construct the lock with a target mutex.

The mutex passed to `sf::Lock` is automatically locked.

**Parameters**

- `mutex` Mutex to lock

**sf::Lock::~Lock ( )**

Destructor.

The destructor of `sf::Lock` automatically unlocks its mutex.

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

- Lock.hpp
sf::MemoryInputStream Class Reference
System module

Implementation of input stream based on a memory chunk. More...

#include <MemoryInputStream.hpp>

Inheritance diagram for sf::MemoryInputStream:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemoryInputStream ()</td>
<td>Default constructor.</td>
<td></td>
</tr>
<tr>
<td>void open (const void *data, std::size_t sizeInBytes)</td>
<td>Open the stream from its data.</td>
<td></td>
</tr>
<tr>
<td>virtual Int64 read (void *data, Int64 size)</td>
<td>Read data from the stream.</td>
<td></td>
</tr>
<tr>
<td>virtual Int64 seek (Int64 position)</td>
<td>Change the current reading position.</td>
<td></td>
</tr>
<tr>
<td>virtual Int64 tell ()</td>
<td>Get the current reading position in the stream.</td>
<td></td>
</tr>
<tr>
<td>virtual Int64 getSize ()</td>
<td>Return the size of the stream.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Implementation of input stream based on a memory chunk.

This class is a specialization of InputStream that reads from data in memory.

It wraps a memory chunk in the common InputStream interface and therefore allows functions that accept such a stream, with content already loaded in memory.

In addition to the virtual functions inherited from InputStream, MemoryInputStream specify the pointer and size of the data in memory.

SFML resource classes can usually be loaded directly from memory, so this class shouldn’t be useful to you unless you create your own algorithms that operate on an InputStream.

Usage example:

```cpp
void process(InputStream& stream);
MemoryInputStream stream;
stream.open(thePtr, theSize);
process(stream);
```

InputStream, FileInputStream

Definition at line 43 of file MemoryInputStream.hpp.
sf::MemoryInputStream::MemoryInputStream ( )

Default constructor.
Member Function Documentation

virtual Int64 sf::MemoryInputStream::getSize ()

Return the size of the stream.

Returns
The total number of bytes available in the stream, or -1 on error

Implements sf::InputStream.

void sf::MemoryInputStream::open ( const void * data,
                                   std::size_t   sizeInBytes )

Open the stream from its data.

Parameters
  data Pointer to the data in memory
  sizeInBytes Size of the data, in bytes

virtual Int64 sf::MemoryInputStream::read ( void * data,
                                            Int64    size )

Read data from the stream.
After reading, the stream's reading position must be advanced by the amount of bytes read.

**Parameters**
- **data** Buffer where to copy the read data
- **size** Desired number of bytes to read

**Returns**
The number of bytes actually read, or -1 on error

Implements `sf::InputStream`.

---

**virtual Int64 sf::MemoryInputStream::seek (Int64 position)**

Change the current reading position.

**Parameters**
- **position** The position to seek to, from the beginning

**Returns**
The position actually sought to, or -1 on error

Implements `sf::InputStream`.

---

**virtual Int64 sf::MemoryInputStream::tell ( )**

Get the current reading position in the stream.

**Returns**
The current position, or -1 on error.

Implements `sf::InputStream`. 
The documentation for this class was generated from the following file:

- MemoryInputStream.hpp

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Mutex Class Reference

System module

Blocks concurrent access to shared resources from multiple threads. More...

```
#include <Mutex.hpp>
```

Inheritance diagram for sf::Mutex:
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutex ()</td>
<td>Default constructor.</td>
<td>More</td>
</tr>
<tr>
<td>~Mutex ()</td>
<td>Destructor.</td>
<td>More</td>
</tr>
<tr>
<td>void lock ()</td>
<td>Lock the mutex.</td>
<td>More</td>
</tr>
<tr>
<td>void unlock ()</td>
<td>Unlock the mutex.</td>
<td>More</td>
</tr>
</tbody>
</table>
Detailed Description

Blocks concurrent access to shared resources from multiple threads.

**Mutex** stands for "MUTual EXclusion".

A mutex is a synchronization object, used when multiple threads are involved.

When you want to protect a part of the code from being accessed simultaneously by multiple threads, you typically use a mutex. When a thread is locked by a mutex, any other thread trying to lock it will be blocked until the mutex is released by the thread that locked it. This way, you can allow only one thread at a time to access a critical region of your code.

Usage example:

```cpp
Database database; // this is a critical resource that needs some protection sf::Mutex mutex;

void thread1()
{
    mutex.lock(); // this call will block the thread if the mutex is already locked
    database.write(...);
    mutex.unlock(); // if thread2 was waiting, it will now be unblocked
}

void thread2()
{
    mutex.lock(); // this call will block the thread if the mutex is already locked
    database.write(...);
    mutex.unlock(); // if thread1 was waiting, it will now be unblocked
}
```

Be very careful with mutexes. A bad usage can lead to bad problems, waiting for each other and the application is globally stuck.

To make the usage of mutexes more robust, particularly in environments where exceptions can be thrown, you should use the helper class `sf::Lock` to lock/unlock mutexes.
SFML mutexes are recursive, which means that you can lock a mutex without creating a deadlock. In this case, the first call to lock() behaves have no effect. However, you must call unlock() exactly as many times as mutex won't be released.

**See also**

sf::Lock

Definition at line 47 of file Mutex.hpp.
Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sf::Mutex::Mutex()</code></td>
<td>Default constructor.</td>
</tr>
<tr>
<td><code>sf::Mutex::~Mutex()</code></td>
<td>Destructor.</td>
</tr>
</tbody>
</table>
Member Function Documentation

**void sf::Mutex::lock ( )**

Lock the mutex.

If the mutex is already locked in another thread, this call will block until released.

**See also**

unlock

**void sf::Mutex::unlock ( )**

Unlock the mutex.

**See also**

lock

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

- Mutex.hpp

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila  ::  Documentation generated by doxygen
Utility class that makes any derived class non-copyable. More...

```cpp
#include <NonCopyable.hpp>
```

Inheritance diagram for `sf::NonCopyable`: 
Protected Member Functions

NonCopyable()
Default constructor. More...
Utility class that makes any derived class non-copyable.

This class makes its instances non-copyable, by explicitly disabling its copy operator.

To create a non-copyable class, simply inherit from `sf::NonCopyable`.

The type of inheritance (public or private) doesn't matter, the copy constructor is declared private in `sf::NonCopyable` so they will end up being inaccessible. This also provides a shorter syntax for inheriting from it (see below).

Usage example:

```cpp
class MyNonCopyableClass : sf::NonCopyable
{
    ...
};
```

Deciding whether the instances of a class can be copied or not is a very important design choice. You are strongly encouraged to think about it before writing a class, and to use `sf::NonCopyable` to prevent many potential future errors when using it. This is also a very important indication to users of your class.

Definition at line 41 of file `NonCopyable.hpp`. 
sf::NonCopyable::NonCopyable ( )

Default constructor.

Because this class has a copy constructor, the compiler will not aut
constructor. That's why we must define it explicitly.

Definition at line 53 of file NonCopyable.hpp.

The documentation for this class was generated from the following file:
- NonCopyable.hpp
sf::String Class Reference

System module

Utility string class that automatically handles conversions between types and encodings.

```
#include <String.hpp>
```
## Public Types

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Type</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef std::basic_string &lt; Uint32 &gt;::iterator</code></td>
<td><strong>Iterator</strong></td>
<td>Iterator type.</td>
<td>More</td>
</tr>
<tr>
<td><code>typedef std::basic_string &lt; Uint32 &gt;::const_iterator</code></td>
<td><strong>ConstIterator</strong></td>
<td>Read-only iterator type.</td>
<td>More</td>
</tr>
</tbody>
</table>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String ()</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>String (char ansiChar, const std::locale &amp;locale=std::locale())</code></td>
<td>Construct from a single ANSI character and a locale.</td>
</tr>
<tr>
<td><code>String (wchar_t wideChar)</code></td>
<td>Construct from single wide character. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>String (Uint32 utf32Char)</code></td>
<td>Construct from single UTF-32 character. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>String (const char *ansiString, const std::locale &amp;locale=std::locale())</code></td>
<td>Construct from a null-terminated C-style ANSI string and a locale.</td>
</tr>
<tr>
<td><code>String (const std::string &amp;ansiString, const std::locale &amp;locale=std::locale())</code></td>
<td>Construct from an ANSI string and a locale.</td>
</tr>
<tr>
<td><code>String (const wchar_t *wideString)</code></td>
<td>Construct from null-terminated C-style wide string.</td>
</tr>
<tr>
<td><code>String (const std::wstring &amp;wideString)</code></td>
<td>Construct from a wide string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>String (const Uint32 *utf32String)</code></td>
<td>Construct from a null-terminated C-style UTF-32 string.</td>
</tr>
<tr>
<td><code>String (const std::basic_string&lt;Uint32&gt; &amp;utf32String)</code></td>
<td>Construct from an UTF-32 string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>String (const String &amp;copy)</code></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>operator std::string () const</code></td>
<td>Implicit conversion operator to std::string (ANSI string)</td>
</tr>
<tr>
<td><code>operator std::wstring () const</code></td>
<td>Implicit conversion operator to std::wstring (wide string)</td>
</tr>
<tr>
<td><code>std::string toAnsiString (const std::locale &amp;locale=std::locale()) const</code></td>
<td>Convert the Unicode string to an ANSI string</td>
</tr>
<tr>
<td><code>std::wstring toWideString () const</code></td>
<td>Convert the Unicode string to a wide string.</td>
</tr>
<tr>
<td><code>std::basic_string&lt; Uint8 &gt; toUtf8 () const</code></td>
<td>Convert the Unicode string to a UTF-8 string</td>
</tr>
<tr>
<td><code>std::basic_string&lt; Uint16 &gt; toUtf16 () const</code></td>
<td>Convert the Unicode string to a UTF-16 string</td>
</tr>
<tr>
<td><code>std::basic_string&lt; Uint32 &gt; toUtf32 () const</code></td>
<td>Convert the Unicode string to a UTF-32 string</td>
</tr>
<tr>
<td><code>String &amp; operator= (const String &amp;right)</code></td>
<td>Overload of assignment operator.</td>
</tr>
<tr>
<td><code>String &amp; operator+= (const String &amp;right)</code></td>
<td>Overload of += operator to append an UTF-32 string</td>
</tr>
<tr>
<td><code>Uint32 operator[] (std::size_t index) const</code></td>
<td>Overload of [] operator to access a character</td>
</tr>
<tr>
<td><code>Uint32 &amp; operator[] (std::size_t index)</code></td>
<td>Overload of [] operator to access a character</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>Clear the string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>getSize()</code></td>
<td>Get the size of the string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>isEmpty()</code></td>
<td>Check whether the string is empty or not. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>erase(position, count)</code></td>
<td>Erase one or more characters from the string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>insert(position, str)</code></td>
<td>Insert one or more characters into the string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>find(str, start)</code></td>
<td>Find a sequence of one or more characters in the string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>replace(position, length, str)</code></td>
<td>Replace a substring with another string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>replace(searchFor, String)</code></td>
<td>Replace all occurrences of a substring with a replacement string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>substring(position, length)</code></td>
<td>Return a part of the string. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>getData()</code></td>
<td>Get a pointer to the C-style array of characters.</td>
</tr>
<tr>
<td><code>begin()</code></td>
<td>Return an iterator to the beginning of the string.</td>
</tr>
<tr>
<td><code>ConstIterator begin()</code></td>
<td>Return an iterator to the beginning of the string.</td>
</tr>
<tr>
<td>Iterator</td>
<td><code>end()</code></td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>Return an iterator to the end of the string.</td>
</tr>
</tbody>
</table>
### Static Public Member Functions

<table>
<thead>
<tr>
<th>Template Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>template&lt;typename T &gt;</code></td>
<td><code>static String fromUtf8 (T begin, T end)</code></td>
<td>Create a new <code>sf::String</code> from a UTF-8 encoded string. More...</td>
</tr>
<tr>
<td><code>template&lt;typename T &gt;</code></td>
<td><code>static String fromUtf16 (T begin, T end)</code></td>
<td>Create a new <code>sf::String</code> from a UTF-16 encoded string. More...</td>
</tr>
<tr>
<td><code>template&lt;typename T &gt;</code></td>
<td><code>static String fromUtf32 (T begin, T end)</code></td>
<td>Create a new <code>sf::String</code> from a UTF-32 encoded string. More...</td>
</tr>
</tbody>
</table>
### Static Public Attributes

<table>
<thead>
<tr>
<th>static const std::size_t</th>
<th>InvalidPos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represents an invalid position in the string.</td>
<td>More</td>
</tr>
</tbody>
</table>
### Friends

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator== (const String &amp;left, const String &amp;right)</code></td>
<td>Equality comparison</td>
</tr>
<tr>
<td><code>bool operator&lt; (const String &amp;left, const String &amp;right)</code></td>
<td>Less than comparison</td>
</tr>
</tbody>
</table>
## Related Functions

(Note that these are not member functions.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool operator==</td>
<td>(const String &amp;left, const String &amp;right)</td>
<td>Overload of == operator to compare two UTF-32 strings.</td>
<td></td>
</tr>
<tr>
<td>bool operator!=</td>
<td>(const String &amp;left, const String &amp;right)</td>
<td>Overload of != operator to compare two UTF-32 strings.</td>
<td></td>
</tr>
<tr>
<td>bool operator&lt;</td>
<td>(const String &amp;left, const String &amp;right)</td>
<td>Overload of &lt; operator to compare two UTF-32 strings.</td>
<td></td>
</tr>
<tr>
<td>bool operator&gt;</td>
<td>(const String &amp;left, const String &amp;right)</td>
<td>Overload of &gt; operator to compare two UTF-32 strings.</td>
<td></td>
</tr>
<tr>
<td>bool operator&lt;=</td>
<td>(const String &amp;left, const String &amp;right)</td>
<td>Overload of &lt;= operator to compare two UTF-32 strings.</td>
<td></td>
</tr>
<tr>
<td>bool operator=&gt;</td>
<td>(const String &amp;left, const String &amp;right)</td>
<td>Overload of &gt;= operator to compare two UTF-32 strings.</td>
<td></td>
</tr>
<tr>
<td>String operator+</td>
<td>(const String &amp;left, const String &amp;right)</td>
<td>Overload of binary + operator to concatenate two strings.</td>
<td></td>
</tr>
</tbody>
</table>
### Detailed Description

Utility string class that automatically handles conversions between types and encodings.

**sf::String** is a utility string class defined mainly for convenience.

It is a Unicode string (implemented using UTF-32), thus it can store any character in the world (European, Chinese, Arabic, Hebrew, etc.).

It automatically handles conversions from/to ANSI and wide strings, so string classes and still be compatible with functions taking a **sf::String**.

```cpp
sf::String s;
std::string s1 = s; // automatically converted to ANSI string
std::wstring s2 = s; // automatically converted to wide string
s = "hello"; // automatically converted from ANSI string
s = L"hello"; // automatically converted from wide string
s += 'a'; // automatically converted from ANSI string
s += L'a'; // automatically converted from wide string
```

Conversions involving ANSI strings use the default user locale. However, it is possible to use a custom locale if necessary:

```cpp
std::locale locale;
sf::String s;
...
std::string s1 = s.to ANSI String(locale);
s = sf::String("hello", locale);
```

**sf::String** defines the most important functions of the standard std::string class: removing, random access, iterating, appending, comparing, etc. However, it is a simple class provided for convenience, and you may have to consider using a more optimized class if your program requires complex string handling. The automatic conversion functions will then take care of converting your string as required.
Please note that SFML also defines a low-level, generic interface for \ classes.

Definition at line 45 of file String.hpp.
Member Typedef Documentation

typedef std::basic_string<Uint32>::const_iterator sf::String::ConstIterator

Read-only iterator type.
Definition at line 53 of file String.hpp.

typedef std::basic_string<Uint32>::iterator sf::String::Iterator

Iterator type.
Definition at line 52 of file String.hpp.
Constructor & Destructor Documentation

sf::String::String ( )

Default constructor.
This constructor creates an empty string.

sf::String::String ( char ansiChar, const std::locale & locale = std::locale() )

Construct from a single ANSI character and a locale.
The source character is converted to UTF-32 according to the given locale.

Parameters
- **ansiChar** ANSI character to convert
- **locale** Locale to use for conversion

sf::String::String ( wchar_t wideChar )

Construct from single wide character.

Parameters
- **wideChar** Wide character to convert
sf::String::String (Uint32 utf32Char)

Construct from single UTF-32 character.

Parameters
  utf32Char UTF-32 character to convert

sf::String::String (const char * ansiString,
const std::locale & locale = std::locale())

Construct from a null-terminated C-style ANSI string and a locale.

The source string is converted to UTF-32 according to the given locale.

Parameters
  ansiString ANSI string to convert
  locale Locale to use for conversion

sf::String::String (const std::string & ansiString,
const std::locale & locale = std::locale())

Construct from an ANSI string and a locale.

The source string is converted to UTF-32 according to the given locale.

Parameters
  ansiString ANSI string to convert
**locale**        Locale to use for conversion

<table>
<thead>
<tr>
<th>sf::String::String ( const wchar_t *  <strong>wideString</strong> )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct from null-terminated C-style wide string.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td><strong>wideString</strong> Wide string to convert</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf::String::String ( const std::wstring &amp;  <strong>wideString</strong> )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct from a wide string.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td><strong>wideString</strong> Wide string to convert</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf::String::String ( const Uint32 *  <strong>utf32String</strong> )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct from a null-terminated C-style UTF-32 string.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td><strong>utf32String</strong> UTF-32 string to assign</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf::String::String ( const std::basic_string&lt; Uint32 &gt; &amp;  <strong>utf32String</strong> )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct from an UTF-32 string.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
</tbody>
</table>
**utf32String** UTF-32 string to assign

**sf::String::String ( const String & copy )**

Copy constructor.

**Parameters**

- **copy** Instance to copy
Member Function Documentation

**Iterator** `sf::String::begin()`

Return an iterator to the beginning of the string.

**Returns**
Read-write iterator to the beginning of the string characters

**See also**
`end`

**ConstIterator** `sf::String::begin()` const

Return an iterator to the beginning of the string.

**Returns**
Read-only iterator to the beginning of the string characters

**See also**
`end`

**void** `sf::String::clear()`

Clear the string.
This function removes all the characters from the string.

**See also**
- isEmpty, erase

### Iterator sf::String::end ( )

Return an iterator to the end of the string.

The end iterator refers to 1 position past the last character; thus it should never be accessed.

**Returns**
- Read-write iterator to the end of the string characters

**See also**
- begin

### ConstIterator sf::String::end ( ) const

Return an iterator to the end of the string.

The end iterator refers to 1 position past the last character; thus it should never be accessed.

**Returns**
- Read-only iterator to the end of the string characters

**See also**
- begin
void sf::String::erase ( std::size_t position,  
               std::size_t count = 1 )  

Erase one or more characters from the string.

This function removes a sequence of count characters starting from position.

Parameters
position Position of the first character to erase
count Number of characters to erase

std::size_t sf::String::find ( const String & str,  
               std::size_t start = 0 )  

Find a sequence of one or more characters in the string.

This function searches for the characters of str in the string, starting from.

Parameters
str Characters to find
start Where to begin searching

Returns
Position of str in the string, or String::InvalidPos if not found

template<typename T >  

static String sf::String::fromUtf16 ( T begin,  


Create a new sf::String from a UTF-16 encoded string.

**Parameters**
- **begin** Forward iterator to the beginning of the UTF-16 sequence
- **end** Forward iterator to the end of the UTF-16 sequence

**Returns**
A sf::String containing the source string

**See also**
fromUtf8, fromUtf32

```cpp
template<typename T>
static String sf::String::fromUtf32 (T begin, T end)
```

Create a new sf::String from a UTF-32 encoded string.

This function is provided for consistency, it is equivalent to using the sf::Uint32* or a std::basic_string<sf::Uint32>.

**Parameters**
- **begin** Forward iterator to the beginning of the UTF-32 sequence
- **end** Forward iterator to the end of the UTF-32 sequence

**Returns**
A sf::String containing the source string
template<typename T >
static String sf::String::fromUtf8 ( T begin,
    T end
)

Create a new sf::String from a UTF-8 encoded string.

**Parameters**

- **begin** Forward iterator to the beginning of the UTF-8 sequence
- **end** Forward iterator to the end of the UTF-8 sequence

**Returns**

A sf::String containing the source string

**See also**

fromUtf8, fromUtf16

---

const Uint32* sf::String::getData ( ) const

Get a pointer to the C-style array of characters.

This functions provides a read-only access to a null-terminated C-style returned pointer is temporary and is meant only for immediate use, thus i

**Returns**

Read-only pointer to the array of characters
std::size_t sf::String::getSize ( ) const

Get the size of the string.

Returns
Number of characters in the string

See also
isEmpty

void sf::String::insert ( std::size_t position, const String & str )

Insert one or more characters into the string.

This function inserts the characters of str into the string, starting from position.

Parameters
position Position of insertion
str Characters to insert

bool sf::String::isEmpty ( ) const

Check whether the string is empty or not.

Returns
True if the string is empty (i.e. contains no character)

See also
clear, getSize
sf::String::operator std::string ( ) const

Implicit conversion operator to std::string (ANSI string)

The current global locale is used for conversion. If you want to specify a locale, use toAnsiString. Characters that do not fit in the target encoding are discarded from the returned string. This operator is defined for convenience, and is equivalent to calling toAnsiString().

Returns
Converted ANSI string

See also
    toAnsiString, operator std::wstring

sf::String::operator std::wstring ( ) const

Implicit conversion operator to std::wstring (wide string)

Characters that do not fit in the target encoding are discarded from the returned string. This operator is defined for convenience, and is equivalent to calling toWideString().

Returns
Converted wide string

See also
    toWideString, operator std::string

String& sf::String::operator+= ( const String & right )
Overload of `+=` operator to append an UTF-32 string.

**Parameters**
- `right` String to append

**Returns**
- Reference to self

```cpp
String& sf::String::operator= ( const String & right )
```

Overload of assignment operator.

**Parameters**
- `right` Instance to assign

**Returns**
- Reference to self

```cpp
Uint32 sf::String::operator[] ( std::size_t index ) const
```

Overload of `[]` operator to access a character by its position.

This function provides read-only access to characters. Note: the behavior is undefined if the range.

**Parameters**
- `index` Index of the character to get

**Returns**
- Character at position `index`
Uint32& sf::String::operator[]( std::size_t index )

Overload of [] operator to access a character by its position.

This function provides read and write access to characters. Note: the behavior is undefined if out of range.

**Parameters**

- `index` Index of the character to get

**Returns**

Reference to the character at position `index`

---

void sf::String::replace ( std::size_t position,
                            std::size_t length,
                            const String & replaceWith )

Replace a substring with another string.

This function replaces the substring that starts at index `position` and string `replaceWith`.

**Parameters**

- `position` Index of the first character to be replaced
- `length` Number of characters to replace. You can pass `InvalidPos` to replace until the end of the string.
- `replaceWith` String that replaces the given substring.
void sf::String::replace ( const String & searchFor, const String & replaceWith )

Replace all occurrences of a substring with a replacement string.

This function replaces all occurrences of searchFor in this string with the

Parameters

- **searchFor** The value being searched for
- **replaceWith** The value that replaces found searchFor values

String sf::String::substring ( std::size_t position, std::size_t length = InvalidPos ) const

Return a part of the string.

This function returns the substring that starts at index position and spans

Parameters

- **position** Index of the first character
- **length** Number of characters to include in the substring (if the string characters as possible are included). InvalidPos can be used the end of the string.

Returns

- String object containing a substring of this object

std::string sf::String::toAnsiString ( const std::locale & locale = std
Convert the Unicode string to an ANSI string.

The UTF-32 string is converted to an ANSI string in the encoding defined by the locale. Characters not fit in the target encoding are discarded from the returned string.

**Parameters**
- `locale` Locale to use for conversion

**Returns**
- Converted ANSI string

**See also**
- `toWideString`, `operator std::string`

```cpp
std::basic_string<Uint16> sf::String::toUtf16() const
```

Convert the Unicode string to a UTF-16 string.

**Returns**
- Converted UTF-16 string

**See also**
- `toUtf8`, `toUtf32`

```cpp
std::basic_string<Uint32> sf::String::toUtf32() const
```

Convert the Unicode string to a UTF-32 string.

This function doesn't perform any conversion, since the string is already stored as UTF-32 internally.
### std::basic_string<Uint8> sf::String::toUtf8 ( ) const

Convert the Unicode string to a UTF-8 string.

**Returns**

Converted UTF-8 string

**See also**

toUtf8, toUtf16

---

### std::wstring sf::String::toWideString ( ) const

Convert the Unicode string to a wide string.

Characters that do not fit in the target encoding are discarded from the returned string.

**Returns**

Converted wide string

**See also**

toAnsiString, operator std::wstring
Friends And Related Function Documentation

bool operator!=(const String & left, const String & right)

Overload of != operator to compare two UTF-32 strings.

Parameters
  - left Left operand (a string)
  - right Right operand (a string)

Returns
  True if both strings are different

String operator+(const String & left, const String & right)

Overload of binary + operator to concatenate two strings.

Parameters
  - left Left operand (a string)
  - right Right operand (a string)

Returns
  Concatenated string
bool operator<( const String & left, const String & right )

Overload of < operator to compare two UTF-32 strings.

Parameters
  left  Left operand (a string)
  right Right operand (a string)

Returns
  True if left is lexicographically before right

bool operator<=( const String & left, const String & right )

Overload of <= operator to compare two UTF-32 strings.

Parameters
  left  Left operand (a string)
  right Right operand (a string)

Returns
  True if left is lexicographically before or equivalent to right

bool operator==( const String & left, const String & right )
Overload of `==` operator to compare two UTF-32 strings.

**Parameters**
- `left` Left operand (a string)
- `right` Right operand (a string)

**Returns**
True if both strings are equal

```cpp
bool operator>( const String & left,
               const String & right )
```

Overload of `>` operator to compare two UTF-32 strings.

**Parameters**
- `left` Left operand (a string)
- `right` Right operand (a string)

**Returns**
True if `left` is lexicographically after `right`

```cpp
bool operator>=( const String & left,
                 const String & right )
```

Overload of `>=` operator to compare two UTF-32 strings.

**Parameters**
- `left` Left operand (a string)
right Right operand (a string)

Returns
True if left is lexicographically after or equivalent to right
## Member Data Documentation

<table>
<thead>
<tr>
<th>const std::size_t sf::String::InvalidPos</th>
</tr>
</thead>
</table>

Represents an invalid position in the string.

Definition at line 58 of file `String.hpp`.

The documentation for this class was generated from the following file:
- `String.hpp`

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila. :: Documentation generated by [doxygen](https://en.wikipedia.org/wiki/Doxygen).
sf::Thread Class Reference
System module

Utility class to manipulate threads. More...

#include <Thread.hpp>

Inheritance diagram for sf::Thread:
### Public Member Functions

- **template<typename F >**
  - **Thread (F function)**
  - Construct the thread from a functor with no argument. [More...](#)

- **template<typename F , typename A >**
  - **Thread (F function, A argument)**
  - Construct the thread from a functor with an argument. [More...](#)

- **template<typename C >**
  - **Thread (void(C::*function)(), C *object)**
  - Construct the thread from a member function and an object. [More..](#)

- **~Thread ()**
  - Destructor. [More...](#)

- **void launch ()**
  - Run the thread. [More...](#)

- **void wait ()**
  - Wait until the thread finishes. [More...](#)

- **void terminate ()**
  - Terminate the thread. [More...](#)
Detailed Description

Utility class to manipulate threads.

Threads provide a way to run multiple parts of the code in parallel.

When you launch a new thread, the execution is split and both the new thread and the caller run in parallel.

To use a sf::Thread, you construct it directly with the function to execute.

sf::Thread has multiple template constructors, which means that you can:

- non-member functions with no argument
- non-member functions with one argument of any type
- functors with no argument (this one is particularly useful for compatibility with std::bind)
- functors with one argument of any type
- member functions from any class with no argument

The function argument, if any, is copied in the sf::Thread instance, corresponding constructor is used). Class instances, however, are pass sure that the object won’t be destroyed while the thread is still using it.

The thread ends when its function is terminated. If the owner sf::Thread thread is finished, the destructor will wait (see wait())

Usage examples:

```cpp
// example 1: non member function with one argument

void threadFunc(int argument)
{
    ... 
}

sf::Thread thread(&threadFunc, 5);
```
thread.launch(); // start the thread (internally calls threadFunc(5))

// example 2: member function

class Task
{
public:
    void run()
    {
        ...
    }
};

Task task;
sf::Thread thread(&Task::run, &task);
thread.launch(); // start the thread (internally calls task.run())

// example 3: functor

struct Task
{
    void operator()()
    {
        ...
    }
};
sf::Thread thread(Task());
thread.launch(); // start the thread (internally calls operator() on the Task instance)

Creating parallel threads of execution can be dangerous: all threads inside the same process share the same memory space, which means that you may end up accessing the same variable from multiple threads at the same time. To prevent this kind of situations, you can use mutexes.

See also

sf::Mutex

Definition at line 48 of file Thread.hpp.
### Constructor & Destructor Documentation

**template<typename F >**

```
sf::Thread::Thread ( F function )
```

Construct the thread from a functor with no argument.

This constructor works for function objects, as well as free functions.

Use this constructor for this kind of function:

```c
void function();
// --- or ----
struct Functor
{
    void operator()();
};
```

Note: this does *not* run the thread, use `launch()`.

**Parameters**

- **function** Functor or free function to use as the entry point of the thread.

**template<typename F , typename A >**

```
sf::Thread::Thread ( F function,
                      A argument )
```

Construct the thread from a functor with an argument.
This constructor works for function objects, as well as free functions. It the argument can have any type (int, std::string, void*, Toto,...).

Use this constructor for this kind of function:

```cpp
void function(int arg);
// --- or ----
struct Functor
{
    void operator()(std::string arg);
};
```

Note: this does not run the thread, use launch().

**Parameters**

- `function` Functor or free function to use as the entry point of the thread
- `argument` argument to forward to the function

Construct the thread from a member function and an object.

This constructor is a template, which means that you can use it with a

```cpp
template<typename C >
sf::Thread::Thread ( void(C::*)(void) function,
                      C * object
                    )
```

```cpp
class MyClass
{
    public:
    void function();
};
```
Note: this does *not* run the thread, use `launch()`.

**Parameters**

- **function** Entry point of the thread
- **object** Pointer to the object to use

**sf::Thread::~Thread ( )**

Destructor.

This destructor calls `wait()`, so that the internal thread cannot survive destroyed.
### Member Function Documentation

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void sf::Thread::launch()</code></td>
<td>Run the thread. This function starts the entry point passed to the thread's constructor, and returns immediately. After this function returns, the thread's function is running in parallel to the calling code.</td>
</tr>
<tr>
<td><code>void sf::Thread::terminate()</code></td>
<td>Terminate the thread. This function immediately stops the thread, without waiting for its function to finish. Terminating a thread with this function is not safe, and can lead to local variables not being destroyed on some operating systems. You should rather try to make the thread function terminate by itself.</td>
</tr>
<tr>
<td><code>void sf::Thread::wait()</code></td>
<td>Wait until the thread finishes. This function will block the execution until the thread's function ends. If the thread's function never ends, the calling thread will block forever. If this function is called without doing anything.</td>
</tr>
</tbody>
</table>

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

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::ThreadLocal Class Reference
System module

Defines variables with thread-local storage. More...

#include <ThreadLocal.hpp>

Inheritance diagram for sf::ThreadLocal:
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ThreadLocal (void *value=NULL)</code></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>~ThreadLocal ()</code></td>
<td>Destructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void setValue (void *value)</code></td>
<td>Set the thread-specific value of the variable. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void * getValue () const</code></td>
<td>Retrieve the thread-specific value of the variable. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Defines variables with thread-local storage.

This class manipulates void* parameters and thus is not appropriate for strongly-typed variables.

You should rather use the `sf::ThreadLocalPtr` template class.

Definition at line 47 of file `ThreadLocal.hpp`. 
sf::ThreadLocal::ThreadLocal ( void * value = NULL )

Default constructor.

**Parameters**

- value Optional value to initialize the variable

sf::ThreadLocal::~ThreadLocal ( )

Destructor.
void* sf::ThreadLocal::getValue ( ) const

Retrieve the thread-specific value of the variable.

Returns
Value of the variable for the current thread

void sf::ThreadLocal::setValue ( void * value )

Set the thread-specific value of the variable.

Parameters
value Value of the variable for the current thread

The documentation for this class was generated from the following file:
- ThreadLocal.hpp
sf::ThreadLocalPtr< T > Class Template Reference

System module

Pointer to a thread-local variable. More...

#include <ThreadLocalPtr.hpp>

Inheritance diagram for sf::ThreadLocalPtr< T >:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ThreadLocalPtr&lt; T &amp; value=NULL&gt;</code></td>
<td>Default constructor.</td>
</tr>
<tr>
<td><code>T &amp; operator* () const</code></td>
<td>Overload of unary operator *.</td>
</tr>
<tr>
<td><code>T * operator-&gt; () const</code></td>
<td>Overload of operator -&gt;</td>
</tr>
<tr>
<td><code>operator T * () const</code></td>
<td>Conversion operator to implicitly convert the pointer to its raw pointer type (T*)</td>
</tr>
<tr>
<td><code>ThreadLocalPtr&lt; T &amp; value&gt;</code></td>
<td>Assignment operator for a raw pointer parameter.</td>
</tr>
<tr>
<td><code>ThreadLocalPtr&lt; T &amp; value&gt;</code></td>
<td>Assignment operator for a <code>ThreadLocalPtr&lt; T &amp; value&gt;</code> parameter.</td>
</tr>
</tbody>
</table>
## Private Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setValue (void *value)</code></td>
<td>Set the thread-specific value of the variable. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void * getValue () const</code></td>
<td>Retrieve the thread-specific value of the variable. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

template<typename T>
class sf::ThreadLocalPtr<T>

Pointer to a thread-local variable.

sf::ThreadLocalPtr is a type-safe wrapper for storing pointers to thread-local variables.

A thread-local variable holds a different value for each different thread, unlike normal variables that are shared.

Its usage is completely transparent, so that it is similar to manipulating the raw pointer directly (like any smart pointer).

Usage example:

```cpp
MyClass object1;
MyClass object2;
sf::ThreadLocalPtr<MyClass> objectPtr;

void thread1()
{
    objectPtr = &object1; // doesn't impact thread2
    ...
}

void thread2()
{
    objectPtr = &object2; // doesn't impact thread1
    ...
}

int main()
{
    // Create and launch the two threads
    sf::Thread t1(&thread1);
    sf::Thread t2(&thread2);
    t1.launch();
    t2.launch();
```
ThreadLocalPtr is designed for internal use; however you can use it implementation.

Definition at line 41 of file ThreadLocalPtr.hpp.
Constructor & Destructor Documentation

template<typename T>

sf::ThreadLocalPtr<T>::ThreadLocalPtr ( T * value = NULL )

Default constructor.

Parameters

value Optional value to initialize the variable
Member Function Documentation

\[
\text{template<typename T>}
\]

\[
\text{sf::ThreadLocalPtr<T>::operator T * ( ) const}
\]

Conversion operator to implicitly convert the pointer to its raw pointer type.

**Returns**

Pointer to the actual object

\[
\text{template<typename T>}
\]

\[
\text{T& sf::ThreadLocalPtr<T>::operator* ( ) const}
\]

Overload of unary operator *.

Like raw pointers, applying the * operator returns a reference to the pointed-to object.

**Returns**

Reference to the thread-local variable

\[
\text{template<typename T>}
\]

\[
\text{T* sf::ThreadLocalPtr<T>::operator-> ( ) const}
\]

Overload of operator ->

Similarly to raw pointers, applying the -> operator returns the pointed-to object.
Returns
   Pointer to the thread-local variable

Assignment operator for a raw pointer parameter.

Parameters
   value Pointer to assign

Returns
   Reference to self

Assignment operator for a ThreadLocalPtr parameter.

Parameters
   right ThreadLocalPtr to assign

Returns
   Reference to self

The documentation for this class was generated from the following file:
   - ThreadLocalPtr.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
sf::Time Class Reference
System module

Represents a time value. More...

```cpp
#include <Time.hpp>
```
### Public Member Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Time()</em></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td>float <em>asSeconds()</em></td>
<td>Return the time value as a number of seconds. More...</td>
</tr>
<tr>
<td>Int32 <em>asMilliseconds()</em></td>
<td>Return the time value as a number of milliseconds. More...</td>
</tr>
<tr>
<td>Int64 <em>asMicroseconds()</em></td>
<td>Return the time value as a number of microseconds. More...</td>
</tr>
</tbody>
</table>
## Static Public Attributes

<table>
<thead>
<tr>
<th>static const Time Zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predefined &quot;zero&quot; time value. More...</td>
</tr>
</tbody>
</table>
## Friends

<table>
<thead>
<tr>
<th>Time</th>
<th>seconds (float)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>milliseconds (Int32)</td>
</tr>
<tr>
<td>Time</td>
<td>microseconds (Int64)</td>
</tr>
</tbody>
</table>
Related Functions

(Note that these are not member functions.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time seconds</strong> (float amount)</td>
<td>Construct a time value from a number of seconds.</td>
</tr>
<tr>
<td><strong>Time milliseconds</strong> (Int32 amount)</td>
<td>Construct a time value from a number of milliseconds.</td>
</tr>
<tr>
<td><strong>Time microseconds</strong> (Int64 amount)</td>
<td>Construct a time value from a number of microseconds.</td>
</tr>
<tr>
<td>bool operator==(Time left, Time right)</td>
<td>Overload of == operator to compare two time values.</td>
</tr>
<tr>
<td>bool operator!=(Time left, Time right)</td>
<td>Overload of != operator to compare two time values.</td>
</tr>
<tr>
<td>bool operator&lt;(Time left, Time right)</td>
<td>Overload of &lt; operator to compare two time values.</td>
</tr>
<tr>
<td>bool operator&gt;(Time left, Time right)</td>
<td>Overload of &gt; operator to compare two time values.</td>
</tr>
<tr>
<td>bool operator&lt;=(Time left, Time right)</td>
<td>Overload of &lt;= operator to compare two time values.</td>
</tr>
<tr>
<td>bool operator&gt;=(Time left, Time right)</td>
<td>Overload of &gt;= operator to compare two time values.</td>
</tr>
<tr>
<td><strong>Time operator-(Time right)</strong></td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>operator+</code></td>
<td>Overload of binary + operator to add two time values.</td>
</tr>
<tr>
<td><code>operator+=</code></td>
<td>Overload of binary += operator to add/assign two time values.</td>
</tr>
<tr>
<td><code>operator-</code></td>
<td>Overload of binary - operator to subtract two time values.</td>
</tr>
<tr>
<td><code>operator-=</code></td>
<td>Overload of binary -= operator to subtract/assign two time values.</td>
</tr>
<tr>
<td><code>operator*</code></td>
<td>Overload of binary * operator to scale a time value.</td>
</tr>
<tr>
<td><code>operator*=</code></td>
<td>Overload of binary *= operator to scale/assign a time value.</td>
</tr>
<tr>
<td><code>operator/</code></td>
<td>Overload of binary / operator to scale a time value.</td>
</tr>
<tr>
<td>Time</td>
<td>operator/ (Time left, Int64 right)</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td></td>
<td>Overload of binary / operator to scale a time value. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time &amp;</th>
<th>operator/= (Time &amp;left, float right)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overload of binary /= operator to scale/assign a time value. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time &amp;</th>
<th>operator/= (Time &amp;left, Int64 right)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overload of binary /= operator to scale/assign a time value. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>float</th>
<th>operator/ (Time left, Time right)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overload of binary / operator to compute the ratio of two time values. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>operator% (Time left, Time right)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overload of binary % operator to compute remainder of a time value. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time &amp;</th>
<th>operator%=(Time &amp;left, Time right)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overload of binary %= operator to compute/assign remainder of a time value. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Represents a time value.

\textit{sf::Time} encapsulates a time value in a flexible way.

It allows to define a time value either as a number of seconds, milliseconds or microseconds.

By using such a flexible interface, the API doesn't impose any fixed type or resolution for time values, and let the user choose its own favorite representation.

Time values support the usual mathematical operations: you can add or subtract times, multiply or divide a time by a number, compare two times, etc.

Since they represent a time span and not an absolute time value, times can be negative.

Usage example:

\begin{verbatim}
\textit{sf::Time} t1 = \textit{sf::seconds}(0.1f);
Int32 milli = t1.\textit{asMilliseconds}(); // 100

\textit{sf::Time} t2 = \textit{sf::milliseconds}(30);
Int64 micro = t2.\textit{asMicroseconds}(); // 30000

\textit{sf::Time} t3 = \textit{sf::microseconds}(-800000);
float sec = t3.\textit{asSeconds}(); // -0.8
\end{verbatim}

\begin{verbatim}
\textbf{void update(\textit{sf::Time} elapsed)}
{
  position += speed * elapsed.\textit{asSeconds}();
}
update(\textit{sf::milliseconds}(100));
\end{verbatim}

\textbf{See also}
sf::Clock

Definition at line 40 of file Time.hpp.
Constructor & Destructor Documentation

sf::Time::Time ( )

Default constructor.

Sets the time value to zero.
## Member Function Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Int64 sf::Time::asMicroseconds ( ) const</strong></td>
<td>Return the time value as a number of microseconds.</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
<td></td>
</tr>
<tr>
<td>Time in microseconds</td>
<td></td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td></td>
</tr>
<tr>
<td>asSeconds, asMilliseconds</td>
<td></td>
</tr>
</tbody>
</table>

| **Int32 sf::Time::asMilliseconds ( ) const** | Return the time value as a number of milliseconds.                         |
| **Returns**                                   |                                                                            |
| Time in milliseconds                          |                                                                            |
| **See also**                                  |                                                                            |
| asSeconds, asMicroseconds                     |                                                                            |

| **float sf::Time::asSeconds ( ) const**      | Return the time value as a number of seconds.                             |
Returns
   Time in seconds

See also
   asMilliseconds, asMicroseconds
## Friends And Related Function Documentation

### Time microseconds ( Int64  amount )

Construct a time value from a number of microseconds.

**Parameters**
- `amount` Number of microseconds

**Returns**
- `Time` value constructed from the amount of microseconds

**See also**
- `seconds`, `milliseconds`

### Time milliseconds ( Int32  amount )

Construct a time value from a number of milliseconds.

**Parameters**
- `amount` Number of milliseconds

**Returns**
- `Time` value constructed from the amount of milliseconds

**See also**
- `seconds`, `microseconds`
```cpp
bool operator!=( Time left,
                Time right )
```

Overload of != operator to compare two time values.

**Parameters**
- **left** Left operand (a time)
- **right** Right operand (a time)

**Returns**
- True if both time values are different

```cpp
Time operator%( Time left,
              Time right )
```

Overload of binary % operator to compute remainder of a time value.

**Parameters**
- **left** Left operand (a time)
- **right** Right operand (a time)

**Returns**
- `left` modulo `right`

```cpp
Time & operator%=( Time & left,
                 Time right )
```
Overload of binary `%=` operator to compute/assign remainder of a time value.

**Parameters**
- `left` Left operand (a time)
- `right` Right operand (a time)

**Returns**
- `left` modulo `right`

**Time operator** (Time `left`,
    float `right`
  )

Overload of binary `*` operator to scale a time value.

**Parameters**
- `left` Left operand (a time)
- `right` Right operand (a number)

**Returns**
- `left` multiplied by `right`

**Time operator** (Time `left`,
    Int64 `right`
  )

Overload of binary `*` operator to scale a time value.

**Parameters**
- `left` Left operand (a time)
**Time operator** *( float  left,  
   
   Time  right  
   )*

Overload of binary * operator to scale a time value.

**Parameters**
- **left**  Left operand (a number)
- **right**  Right operand (a time)

**Returns**
*left* multiplied by *right*

---

**Time operator** *( Int64  left,  
   
   Time  right  
   )*

Overload of binary * operator to scale a time value.

**Parameters**
- **left**  Left operand (a number)
- **right**  Right operand (a time)

**Returns**
*left* multiplied by *right*
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Parameters</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time &amp; operator*=( Time &amp; left, float right )</td>
<td>Overload of binary *= operator to scale/assign a time value.</td>
<td>left Left operand (a time) right Right operand (a number)</td>
<td>left multiplied by right</td>
</tr>
<tr>
<td>Time &amp; operator*=( Time &amp; left, Int64 right )</td>
<td>Overload of binary *= operator to scale/assign a time value.</td>
<td>left Left operand (a time) right Right operand (a number)</td>
<td>left multiplied by right</td>
</tr>
<tr>
<td>Time operator+ ( Time left, Time right )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overload of binary + operator to add two time values.

**Parameters**
- **left** Left operand (a time)
- **right** Right operand (a time)

**Returns**
- Sum of the two times values

```
Time & operator+= ( Time & left,
                 Time   right
             )
```

Overload of binary += operator to add/assign two time values.

**Parameters**
- **left** Left operand (a time)
- **right** Right operand (a time)

**Returns**
- Sum of the two times values

```
Time operator- ( Time   right )
```

Overload of unary - operator to negate a time value.

**Parameters**
- **right** Right operand (a time)

**Returns**
Opposite of the time value

**Time operator-** (Time left, Time right)

Overload of binary - operator to subtract two time values.

**Parameters**
- left Left operand (a time)
- right Right operand (a time)

**Returns**
- Difference of the two times values

**Time & operator-**= (Time & left, Time right)

Overload of binary -= operator to subtract/assign two time values.

**Parameters**
- left Left operand (a time)
- right Right operand (a time)

**Returns**
- Difference of the two times values

**Time operator/** (Time left, Time right)

Overload of binary / operator to divide two time values.

**Parameters**
- left Left operand (a time)
- right Right operand (a time)

**Returns**
- Quotient of the two times values

**Time operator%** (Time left, Time right)

Overload of binary % operator to find the remainder of two time values.

**Parameters**
- left Left operand (a time)
- right Right operand (a time)

**Returns**
- Remainder of the two times values
Overload of binary / operator to scale a time value.

**Parameters**

- `left` Left operand (a time)
- `right` Right operand (a number)

**Returns**

`left` divided by `right`
### Parameters
- **left** Left operand (a time)
- **right** Right operand (a time)

### Returns
- *left* divided by *right*

```c
Time & operator/= ( Time & left,
    float     right
)
```

Overload of binary /= operator to scale/assign a time value.

### Parameters
- **left** Left operand (a time)
- **right** Right operand (a number)

### Returns
- *left* divided by *right*

```c
Time & operator/= ( Time & left,
    Int64   right
)
```

Overload of binary /= operator to scale/assign a time value.
bool operator< ( Time left,  
    Time right  
) 

Overload of < operator to compare two time values.

Parameters

left  Left operand (a time)
right Right operand (a time)

Returns

True if left is lesser than right

bool operator<= ( Time left,  
    Time right  
) 

Overload of <= operator to compare two time values.

Parameters

left  Left operand (a time)
right Right operand (a time)

Returns

True if left is lesser or equal than right

bool operator== ( Time left,  

Overload of `==` operator to compare two time values.

**Parameters**
- `left` Left operand (a time)
- `right` Right operand (a time)

**Returns**
- True if both time values are equal

---

```cpp
bool operator>( Time left,
               Time right
 )
```

Overload of `>` operator to compare two time values.

**Parameters**
- `left` Left operand (a time)
- `right` Right operand (a time)

**Returns**
- True if `left` is greater than `right`

---

```cpp
bool operator>=( Time left,
                 Time right
 )
```

Overload of `>=` operator to compare two time values.
Parameters
   left  Left operand (a time)
   right Right operand (a time)

Returns
   True if left is greater or equal than right

---

**Time** seconds (float amount)

Construct a time value from a number of seconds.

Parameters
   amount Number of seconds

Returns
   Time value constructed from the amount of seconds

See also
   milliseconds, microseconds
Member Data Documentation

### `const Time sf::Time::Zero`

Predefined "zero" time value.

Definition at line 85 of file `Time.hpp`.

The documentation for this class was generated from the following file:
- `Time.hpp`
sf::Utf< N > Class Template Reference
System module

Utility class providing generic functions for UTF conversions. More...

#include <Utf.hpp>
Detailed Description

template<unsigned int N>
class sf::Utf< N >

Utility class providing generic functions for UTF conversions.

sf::Utf is a low-level, generic interface for counting, iterating, encoding and strings. It is able to handle ANSI, wide, latin-1, UTF-8, UTF-16 and UTF-32 encodings.

sf::Utf<X> functions are all static, these classes are not meant to be instantiated. All the functions are template, so that you can use any character / string type for a given encoding.

It has 3 specializations:

- sf::Utf<8> (typedef'd to sf::Utf8)
- sf::Utf<16> (typedef'd to sf::Utf16)
- sf::Utf<32> (typedef'd to sf::Utf32)

Definition at line 41 of file Utf.hpp.

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

- Utf.hpp
sf::Vector2< T > Class Template Reference

System module

Utility template class for manipulating 2-dimensional vectors. More...

#include <Vector2.hpp>
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vector2 ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Vector2 (T X, T Y)</strong></td>
<td>Construct the vector from its coordinates. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>template&lt;typename U &gt; Vector2 (const Vector2&lt; U &gt; &amp;vector)</strong></td>
<td>Construct the vector from another type of vector. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Public Attributes

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T</strong></td>
<td><strong>x</strong></td>
</tr>
<tr>
<td></td>
<td>X coordinate of the vector. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T</strong></td>
<td><strong>y</strong></td>
</tr>
<tr>
<td></td>
<td>Y coordinate of the vector. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Related Functions

(Note that these are not member functions.)

\begin{verbatim}
template<typename T >
Vector2< T >  operator- (const Vector2< T > &right)
Overload of unary operator -. More...
\end{verbatim}

\begin{verbatim}
template<typename T >
Vector2< T > &  operator+= (Vector2< T > &left, const Vector2< T > &right)
Overload of binary operator +=. More...
\end{verbatim}

\begin{verbatim}
template<typename T >
Vector2< T > &  operator-= (Vector2< T > &left, const Vector2< T > &right)
Overload of binary operator -=. More...
\end{verbatim}

\begin{verbatim}
template<typename T >
Vector2< T >  operator+ (const Vector2< T > &left, const Vector2< T > &right)
Overload of binary operator +. More...
\end{verbatim}

\begin{verbatim}
template<typename T >
Vector2< T >  operator- (const Vector2< T > &left, const Vector2< T > &right)
Overload of binary operator -. More...
\end{verbatim}

\begin{verbatim}
template<typename T >
Vector2< T >  operator* (const Vector2< T > &left, const Vector2< T > &right)
Overload of binary operator *. More...
\end{verbatim}

\begin{verbatim}
template<typename T >
Vector2< T >  operator* (T left, const Vector2< T > &right)
Overload of binary operator *. More...
\end{verbatim}

\begin{verbatim}
template<typename T >
Vector2< T > &  operator*= (Vector2< T > &left, T right)
Overload of binary operator *=. More...
\end{verbatim}
<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>template&lt;typename T &gt;</code></td>
<td>Overload of binary operator <code>/</code>. More...</td>
</tr>
<tr>
<td><code>Vector2&lt; T &gt;</code></td>
<td>Overload of binary operator <code>/</code>. More...</td>
</tr>
<tr>
<td><code>template&lt;typename T &gt;</code></td>
<td>Overload of binary operator <code>//=</code>. More...</td>
</tr>
<tr>
<td><code>bool</code></td>
<td>Overload of binary operator <code>==</code>. More...</td>
</tr>
<tr>
<td><code>bool</code></td>
<td>Overload of binary operator <code>!=</code>. More...</td>
</tr>
</tbody>
</table>
Detailed Description

template<typename T>
class sf::Vector2< T >

Utility template class for manipulating 2-dimensional vectors.

sf::Vector2 is a simple class that defines a mathematical vector with two c

It can be used to represent anything that has two dimensions: a size, a po

The template parameter T is the type of the coordinates. It can be a
operations (+, -, /, *) and comparisons (==, !=), for example int or float.

You generally don't have to care about the templated form (sf::\nspecializations have special typedefs:

- sf::Vector2<float> is sf::Vector2f
- sf::Vector2<int> is sf::Vector2i
- sf::Vector2<unsigned int> is sf::Vector2u

The sf::Vector2 class has a small and simple interface, its x and y me
(there are no accessors like setX(), getX()) and it contains no mathematic
product, length, etc.

Usage example:

```cpp
sf::Vector2f v1(16.5f, 24.f);
v1.x = 18.2f;
float y = v1.y;
sf::Vector2f v2 = v1 * 5.f;
sf::Vector2f v3;
```
v3 = v1 + v2;
bool different = (v2 != v3);

Note: for 3-dimensional vectors, see sf::Vector3.

Definition at line 37 of file Vector2.hpp.
## Constructor & Destructor Documentation

### Default constructor.

*Creates a `Vector2(0, 0)`.*

```cpp
template<typename T>
sf::Vector2< T >::Vector2()
```

### Construct the vector from its coordinates.

**Parameters**

- `X`: X coordinate
- `Y`: Y coordinate

```cpp
template<typename T>
sf::Vector2< T >::Vector2(T X, T Y)
```

### Construct the vector from another type of vector.

```cpp
template<typename T>
template<typename U>
sf::Vector2< T >::Vector2(const Vector2< U >& vector)
```

*Constructs the vector from another type of vector.*
This constructor doesn't replace the copy constructor, it's called only when \( U \neq T \). A call to this constructor will fail to compile if \( U \) is not convertible to \( T \).

**Parameters**

- **vector** Vector to convert
template<typename T >

bool operator!= ( const Vector2< T > & left,
                 const Vector2< T > & right )

Overload of binary operator !=.
This operator compares strict difference between two vectors.

Parameters
  left  Left operand (a vector)
  right Right operand (a vector)

Returns
  True if left is not equal to right

template<typename T >

Vector2< T > operator* ( const Vector2< T > & left,
                         T                    right )

Overload of binary operator *.

Parameters
  left  Left operand (a vector)
  right Right operand (a scalar value)
template<typename T >

Vector2< T > operator* ( T left, const Vector2< T > & right )

Overload of binary operator *.

Parameters

left  Left operand (a scalar value)
right Right operand (a vector)

Returns

Memberwise multiplication by left

template<typename T >

Vector2< T > & operator*=( Vector2< T > & left, T right )

Overload of binary operator *=.

This operator performs a memberwise multiplication by right, and assign:

Parameters

left  Left operand (a vector)
right Right operand (a scalar value)
Returns

Reference to \textit{left}

\begin{verbatim}
template<typename T >
Vector2< T > operator+ ( const Vector2< T > & left,
        const Vector2< T > & right
    )
\end{verbatim}

Overload of binary operator +.

\textbf{Parameters}

- \textit{left}  Left operand (a vector)
- \textit{right} Right operand (a vector)

\textbf{Returns}

Memberwise addition of both vectors

\begin{verbatim}
template<typename T >
Vector2< T > & operator+=( Vector2< T > & left,
        const Vector2< T > & right
    )
\end{verbatim}

Overload of binary operator +=.

This operator performs a memberwise addition of both vectors, and assi\textit{...}
template<typename T >

**Vector2< T > operator- ( const Vector2< T > & right )**

Overload of unary operator -.  

**Parameters**  
right Vector to negate

**Returns**  
Memberwise opposite of the vector

---

template<typename T >

**Vector2< T > operator- ( const Vector2< T > & left,  
                       const Vector2< T > & right )**

Overload of binary operator -.  

**Parameters**  
left Left operand (a vector)  
right Right operand (a vector)

**Returns**  
Memberwise subtraction of both vectors

---

template<typename T >

**Vector2< T > & operator-= ( Vector2< T > & left,  
                           Vector2< T > & right )**
Overload of binary operator \(-=\).

This operator performs a memberwise subtraction of both vectors, and assigns the result to \( \text{left} \).

**Parameters**
- \( \text{left} \): Left operand (a vector)
- \( \text{right} \): Right operand (a vector)

**Returns**
Reference to \( \text{left} \)

Overload of binary operator \(/\).

**Parameters**
- \( \text{left} \): Left operand (a vector)
- \( \text{right} \): Right operand (a scalar value)

**Returns**
Memberwise division by \( \text{right} \)

Overload of memberwise division operator \(-/=\)
Overload of binary operator /=.

This operator performs a memberwise division by right, and assigns the result to left.

**Parameters**

- **left**  Left operand (a vector)
- **right** Right operand (a scalar value)

**Returns**

Reference to left

```cpp
template<typename T>
bool operator== ( const Vector2< T > & left, 
                 const Vector2< T > & right )
```

Overload of binary operator ==.

This operator compares strict equality between two vectors.

**Parameters**

- **left**  Left operand (a vector)
- **right** Right operand (a vector)

**Returns**

True if left is equal to right
Member Data Documentation

```
template<typename T>
T sf::Vector2< T >::x
```

X coordinate of the vector.

Definition at line 75 of file Vector2.hpp.

```
template<typename T>
T sf::Vector2< T >::y
```

Y coordinate of the vector.

Definition at line 76 of file Vector2.hpp.

The documentation for this class was generated from the following file:
- Vector2.hpp
sf::Vector3< T > Class Template Reference

System module

Utility template class for manipulating 3-dimensional vectors. More...

#include <Vector3.hpp>
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Vector3()</code></td>
<td>Default constructor.</td>
<td>More...</td>
</tr>
<tr>
<td><code>Vector3(T X, T Y, T Z)</code></td>
<td>Construct the vector from its coordinates.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename U&gt; Vector3(const Vector3&lt;U&gt; &amp;vector)</code></td>
<td>Construct the vector from another type of vector.</td>
<td>More...</td>
</tr>
</tbody>
</table>
## Public Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>T x</td>
<td>X coordinate of the vector.</td>
<td></td>
</tr>
<tr>
<td>T y</td>
<td>Y coordinate of the vector.</td>
<td></td>
</tr>
<tr>
<td>T z</td>
<td>Z coordinate of the vector.</td>
<td></td>
</tr>
</tbody>
</table>
Related Functions

(Note that these are not member functions.)

<table>
<thead>
<tr>
<th>Template</th>
<th>Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; operator- (const Vector3&lt;T&gt; &amp;left)</code></td>
<td>Overload of unary operator <code>-</code>.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; &amp; operator+= (Vector3&lt;T&gt; &amp;left, const Vector3&lt;T&gt; &amp;right)</code></td>
<td>Overload of binary operator <code>+=</code>.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; &amp; operator-= (Vector3&lt;T&gt; &amp;left, const Vector3&lt;T&gt; &amp;right)</code></td>
<td>Overload of binary operator <code>-=</code>.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; operator+ (const Vector3&lt;T&gt; &amp;left, const Vector3&lt;T&gt; &amp;right)</code></td>
<td>Overload of binary operator <code>+</code>.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; operator- (const Vector3&lt;T&gt; &amp;left, const Vector3&lt;T&gt; &amp;right)</code></td>
<td>Overload of binary operator <code>-</code>.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; operator* (const Vector3&lt;T&gt; &amp;left, T right)</code></td>
<td>Overload of binary operator <code>*</code>.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; operator* (T left, const Vector3&lt;T&gt; &amp;right)</code></td>
<td>Overload of binary operator <code>*</code>.</td>
<td>More...</td>
</tr>
<tr>
<td><code>template&lt;typename T&gt;</code></td>
<td><code>Vector3&lt;T&gt; &amp; operator*= (Vector3&lt;T&gt; &amp;left, T right)</code></td>
<td>Overload of binary operator <code>*=</code>.</td>
<td>More...</td>
</tr>
<tr>
<td>Template</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| template<typename T >
  Vector3<T>  operator/ (const Vector3<T> &left, T right)
  Overload of binary operator /.
  More... |
| template<typename T >
  Vector3<T> &  operator/= (Vector3<T> &left, T right)
  Overload of binary operator /=. More... |
| template<typename T >
  bool  operator==(const Vector3<T> &left, const Vector3<T> &right)
  Overload of binary operator ==.
  More... |
| template<typename T >
  bool  operator!=(const Vector3<T> &left, const Vector3<T> &right)
  Overload of binary operator !=.
  More... |
Detailed Description

template<typename T>
class sf::Vector3< T >

Utility template class for manipulating 3-dimensional vectors.

sf::Vector3 is a simple class that defines a mathematical vector with three
It can be used to represent anything that has three dimensions: a size, a π
The template parameter T is the type of the coordinates. It can be a
operations (+, -, /, *) and comparisons (==, !=), for example int or float.
You generally don’t have to care about the templated form (sf::
v specializations have special typedefs:

- sf::Vector3<float> is sf::Vector3f
- sf::Vector3<int> is sf::Vector3i

The sf::Vector3 class has a small and simple interface, its x and y members
(there are no accessors like setX(), getX()) and it contains no mathematical
product, length, etc.

Usage example:

```cpp
sf::Vector3f v1(16.5f, 24.f, -8.2f);
v1.x = 18.2f;
float y = v1.y;
float z = v1.z;
sf::Vector3f v2 = v1 * 5.f;
sf::Vector3f v3;
v3 = v1 + v2;
```
bool different = (v2 != v3);

Note: for 2-dimensional vectors, see sf::Vector2.

Definition at line 37 of file Vector3.hpp.
Constructor & Destructor Documentation

```cpp
template<typename T>
sf::Vector3<T>::Vector3 ()
```

Default constructor.

Creates a Vector3(0, 0, 0).

```cpp
template<typename T>
sf::Vector3<T>::Vector3 ( T X,
                             T Y,
                             T Z )
```

Construct the vector from its coordinates.

**Parameters**
- X X coordinate
- Y Y coordinate
- Z Z coordinate

```cpp
template<typename T>
template<typename U >
sf::Vector3<T>::Vector3 ( const Vector3<U> & vector )
```
Construct the vector from another type of vector.

This constructor doesn't replace the copy constructor, it's called only when U != T. A call to this constructor will fail to compile if U is not convertible to T.

**Parameters**

- **vector** Vector to convert
**Friends And Related Function Documentation**

```cpp
template<typename T >
bool operator!= ( const Vector3< T > & left,
                 const Vector3< T > & right )
```

Overload of binary operator `!=`.
This operator compares strict difference between two vectors.

**Parameters**
- `left`  Left operand (a vector)
- `right` Right operand (a vector)

**Returns**
- True if `left` is not equal to `right`

```cpp
template<typename T >
Vector3< T > operator* ( const Vector3< T > & left,
                         T                        right )
```

Overload of binary operator `*`.

**Parameters**
- `left`  Left operand (a vector)
- `right` Right operand (a scalar value)
template<typename T >

Vector3< T > operator* ( T left, const Vector3< T > & right )

Overload of binary operator *.

Parameters
left Left operand (a scalar value)
right Right operand (a vector)

Returns
Memberwise multiplication by left

template<typename T >

Vector3< T > & operator*=( Vector3< T > & left, T right )

Overload of binary operator *=.  

This operator performs a memberwise multiplication by right, and assigns:

Parameters
left Left operand (a vector)
right Right operand (a scalar value)
Returns
Reference to \textit{left}

\begin{verbatim}
template<typename T >
Vector3< T > operator+ ( const Vector3< T >& left,
                       const Vector3< T >& right
                     )
\end{verbatim}

Overload of binary operator \texttt{+}.

**Parameters**

- \texttt{left}  Left operand (a vector)
- \texttt{right} Right operand (a vector)

**Returns**

Memberwise addition of both vectors

\begin{verbatim}
template<typename T >
Vector3< T >& operator+=( Vector3< T >& left,
                         const Vector3< T >& right
                       )
\end{verbatim}

Overload of binary operator \texttt{+=}.

This operator performs a memberwise addition of both vectors, and assign

**Parameters**

- \texttt{left}  Left operand (a vector)
- \texttt{right} Right operand (a vector)

**Returns**
template<typename T >

Vector3< T > operator- ( const Vector3< T > & left )

Overload of unary operator -.

Parameters

left Vector to negate

Returns

Memberwise opposite of the vector

template<typename T >

Vector3< T > operator- ( const Vector3< T > & left,
const Vector3< T > & right )

Overload of binary operator -.

Parameters

left Left operand (a vector)
right Right operand (a vector)

Returns

Memberwise subtraction of both vectors

template<typename T >

Vector3< T > & operator-= ( Vector3< T > & left,
Overload of binary operator `-=`.

This operator performs a memberwise subtraction of both vectors, and assigns the result to the left operand.

**Parameters**
- `left`  Left operand (a vector)
- `right` Right operand (a vector)

**Returns**
Reference to `left`

---

Overload of binary operator `/`.

**Parameters**
- `left`  Left operand (a vector)
- `right` Right operand (a scalar value)

**Returns**
Memberwise division by `right`
Overload of binary operator /=.

This operator performs a memberwise division by right, and assigns the result to left.

**Parameters**
- **left** Left operand (a vector)
- **right** Right operand (a scalar value)

**Returns**
Reference to left

```cpp
template<typename T>
bool operator== ( const Vector3<T> & left, const Vector3<T> & right )
```

Overload of binary operator ==.

This operator compares strict equality between two vectors.

**Parameters**
- **left** Left operand (a vector)
- **right** Right operand (a vector)

**Returns**
True if left is equal to right
The document for this class was generated from the following file:

- Vector3.hpp
Window module

Provides OpenGL-based windows, and abstractions for events and input.
# Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sf::Context</code></td>
<td>Class holding a valid drawing context. More...</td>
</tr>
<tr>
<td><code>sf::ContextSettings</code></td>
<td>Structure defining the settings of the OpenGL context attached to a window.</td>
</tr>
<tr>
<td><code>sf::Event</code></td>
<td>Defines a system event and its parameters. More...</td>
</tr>
<tr>
<td><code>sf::GlResource</code></td>
<td>Base class for classes that require an OpenGL context. More...</td>
</tr>
<tr>
<td><code>sf::Joystick</code></td>
<td>Give access to the real-time state of the joysticks. More...</td>
</tr>
<tr>
<td><code>sf::Keyboard</code></td>
<td>Give access to the real-time state of the keyboard. More...</td>
</tr>
<tr>
<td><code>sf::Mouse</code></td>
<td>Give access to the real-time state of the mouse. More...</td>
</tr>
<tr>
<td><code>sf::Sensor</code></td>
<td>Give access to the real-time state of the sensors. More...</td>
</tr>
<tr>
<td><code>sf::Touch</code></td>
<td>Give access to the real-time state of the touches. More...</td>
</tr>
<tr>
<td><code>sf::VideoMode</code></td>
<td>VideoMode defines a video mode (width, height, bpp) More...</td>
</tr>
<tr>
<td><code>sf::Window</code></td>
<td></td>
</tr>
</tbody>
</table>
Window that serves as a target for OpenGL rendering. More...
## Typedefs

typedef platform–specific `sf::WindowHandle`

Define a low-level window handle type, specific to each platform.
## Enumerations

```
enum {
    sf::Style::None = 0, sf::Style::Titlebar = 1 << 0, sf::Style::Resize = 2,
    sf::Style::Fullscreen = 1 << 3, sf::Style::Default = Titlebar |Resize
}
```

Enumeration of the window styles. [More...](#)
Detailed Description

Provides OpenGL-based windows, and abstractions for events and input.
Typedef Documentation

**sf::WindowHandle**

Define a low-level window handle type, specific to each platform.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>HWND</td>
</tr>
<tr>
<td>Linux/FreeBSD</td>
<td>Window</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>either NSWindow* or NSView*, disguised as void*</td>
</tr>
<tr>
<td>iOS</td>
<td>UIWindow*</td>
</tr>
<tr>
<td>Android</td>
<td>ANativeWindow*</td>
</tr>
</tbody>
</table>

**Mac OS X Specification**

On Mac OS X, a `sf::Window` can be created either from an existing `NSWindow` window. If a `sf::Window` is created from a window, SFML will use its content. The `sf::Window::getSystemHandle()` method will return the handle that was used to create the window, which is a `NSWindow*` by default.

Definition at line 68 of file `WindowHandle.hpp`. 
## Enumeration Type Documentation

**anonymous enum**

Enumeration of the window styles.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No border / title bar (this flag and all others are mutually exclusive)</td>
</tr>
<tr>
<td>Titlebar</td>
<td>Title bar + fixed border.</td>
</tr>
<tr>
<td>Resize</td>
<td>Title bar + resizable border + maximize button.</td>
</tr>
<tr>
<td>Close</td>
<td>Title bar + close button.</td>
</tr>
<tr>
<td>Fullscreen</td>
<td>Fullscreen mode (this flag and all others are mutually exclusive)</td>
</tr>
<tr>
<td>Default</td>
<td>Default window style.</td>
</tr>
</tbody>
</table>

Definition at line 38 of file WindowStyle.hpp.

SFML is licensed under the terms and conditions of the zlib/png license.
sf::Context Class Reference

Window module

Class holding a valid drawing context. More...

#include <Context.hpp>

Inheritance diagram for sf::Context:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Context ()</code></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td><code>~Context ()</code></td>
<td>Destructor. More...</td>
</tr>
<tr>
<td><code>bool setActive (bool active)</code></td>
<td>Activate or deactivate explicitly the context. More...</td>
</tr>
<tr>
<td><code>const ContextSettings &amp; getSettings () const</code></td>
<td>Get the settings of the context. More...</td>
</tr>
<tr>
<td><code>Context (const ContextSettings &amp;settings, unsigned int width, unsigned int height)</code></td>
<td>Construct a in-memory context. More...</td>
</tr>
</tbody>
</table>
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static bool</td>
<td>isExtensionAvailable</td>
<td>Check whether a given OpenGL extension is available.</td>
</tr>
<tr>
<td>static GLFunctionPointer</td>
<td>getFunction</td>
<td>Get the address of an OpenGL function.</td>
</tr>
<tr>
<td>static const Context *</td>
<td>getActiveContext</td>
<td>Get the currently active context.</td>
</tr>
</tbody>
</table>
Static Private Member Functions

```cpp
static void ensureGLContext ()
Empty function for ABI compatibility, use acquireTransientCor
```
Class holding a valid drawing context.

If you need to make OpenGL calls without having an active window (like in a thread), you can use an instance of this class to get a valid context.

Having a valid context is necessary for every OpenGL call.

Note that a context is only active in its current thread, if you create a new thread it will have no valid context by default.

To use a sf::Context instance, just construct it and let it live as long as you need a valid context. No explicit activation is needed, all it has to do is to exist. Its destructor will take care of deactivating and freeing all the attached resources.

Usage example:

```cpp
void threadFunction(void*)
{
    sf::Context context;
    // from now on, you have a valid context

    // you can make OpenGL calls
    glClear(GL_DEPTH_BUFFER_BIT);
}
// the context is automatically deactivated and destroyed
// by the sf::Context destructor
```

Definition at line 50 of file Context.hpp.
Constructor & Destructor Documentation

sf::Context::Context ( )

Default constructor.

The constructor creates and activates the context

sf::Context::~Context ( )

Destructor.

The destructor deactivates and destroys the context

sf::Context::Context ( const ContextSettings & settings, unsigned int width, unsigned int height )

Construct a in-memory context.

This constructor is for internal use, you don't need to bother with it.

Parameters

- settings Creation parameters
- width Back buffer width
height  Back buffer height
### static const Context* sf::Context::getActiveContext()

Get the currently active context.

**Returns**

The currently active context or NULL if none is active.

### static GLFunctionPointer sf::Context::getFunction (const char * name)

Get the address of an OpenGL function.

**Parameters**

- `name`: Name of the function to get the address of

**Returns**

Address of the OpenGL function, 0 on failure.

### const ContextSettings& sf::Context::getSettings()

Get the settings of the context.

Note that these settings may be different than the ones passed to `ContextSettings` constructor; they are indeed adjusted if the original settings are not directly supported by the system.
**Returns**

Structure containing the settings

---

```cpp
static bool sf::Context::isExtensionAvailable ( const char * name )
```

Check whether a given OpenGL extension is available.

**Parameters**

- `name` Name of the extension to check for

**Returns**

True if available, false if unavailable

---

```cpp
bool sf::Context::setActive ( bool active )
```

Activate or deactivate explicitly the context.

**Parameters**

- `active` True to activate, false to deactivate

**Returns**

True on success, false on failure

---

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

- `Context.hpp`

---

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::ContextSettings Class Reference

Window module

Structure defining the settings of the OpenGL context attached to a window.

```cpp
#include <ContextSettings.hpp>
```
## Public Types

<table>
<thead>
<tr>
<th>enum Attribute { Default = 0, Core = 1 &lt;&lt; 0, Debug = 1 &lt;&lt; 2 }</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration of the context attribute flags. More...</td>
</tr>
</tbody>
</table>
Public Member Functions

**ContextSettings** (unsigned int depth=0, unsigned int stencil=0, unsigned int major=1, unsigned int minor=1, unsigned int attributes=Default, bool sfl)

Default constructor. More...
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>depthBits</td>
<td>Bits of the depth buffer.</td>
<td>More</td>
</tr>
<tr>
<td>unsigned int</td>
<td>stencilBits</td>
<td>Bits of the stencil buffer.</td>
<td>More</td>
</tr>
<tr>
<td>unsigned int</td>
<td>antialiasingLevel</td>
<td>Level of antialiasing.</td>
<td>More</td>
</tr>
<tr>
<td>unsigned int</td>
<td>majorVersion</td>
<td>Major number of the context version to create.</td>
<td>More</td>
</tr>
<tr>
<td>unsigned int</td>
<td>minorVersion</td>
<td>Minor number of the context version to create.</td>
<td>More</td>
</tr>
<tr>
<td>Uint32</td>
<td>attributeFlags</td>
<td>The attribute flags to create the context with.</td>
<td>More</td>
</tr>
<tr>
<td>bool</td>
<td>sRgbCapable</td>
<td>Whether the context framebuffer is sRGB capable.</td>
<td>More</td>
</tr>
</tbody>
</table>
Detailed Description

Structure defining the settings of the OpenGL context attached to a window.

**ContextSettings** allows to define several advanced settings of the OpenGL context attached to a window.

All these settings with the exception of the compatibility flag and anti-aliasing level have no impact on the regular SFML rendering (graphics module), so you may need to use this structure as a windowing system for custom OpenGL rendering.

The depthBits and stencilBits members define the number of bits per pixel depth and stencil buffers.

antialiasingLevel represents the requested number of multisampling levels.

majorVersion and minorVersion define the version of the OpenGL context that you want. Only versions greater or equal to 3.0 are relevant; versions lesser than 3.0 are all handled the same way (i.e. you can use any version < 3.0 if you don’t want an OpenGL 3 context).

When requesting a context with a version greater or equal to 3.2, you have the option of specifying whether the context should follow the core or compatibility profile of all newer (>= 3.2) OpenGL specifications. For versions 3.0 and 3.1 there is only the core profile. By default a compatibility context is created. You only need to specify the core flag if you want a core profile context to use with your own OpenGL rendering.

**Warning:** The graphics module will not function if you request a core profile context. Make sure the attributes are set to Default if you want to use the graphics module.

Setting the debug attribute flag will request a context with additional debugging features enabled. Depending on the system, this might be required for advanced OpenGL debugging. OpenGL debugging is disabled by default.

**Special Note for OS X:** Apple only supports choosing between either a compatibility or core profile OpenGL context.
core context (OpenGL version depends on the operating system version
contexts are not supported. Further information is available on the OpenGL
also currently does not support debug contexts.

Please note that these values are only a hint. No failure will be reported if
not supported by the system; instead, SFML will try to find the closest value
the settings that the window actually used to create its context, with Window::getSettings()

Definition at line 36 of file ContextSettings.hpp.
# Member Enumeration Documentation

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Non-debug, compatibility context (this and the core attribute are mutually exclusive)</td>
</tr>
<tr>
<td>Core</td>
<td>Core attribute.</td>
</tr>
<tr>
<td>Debug</td>
<td>Debug attribute.</td>
</tr>
</tbody>
</table>

Definition at line 42 of file `ContextSettings.hpp`.
sf::ContextSettings::ContextSettings (unsigned int depth = 0, unsigned int stencil = 0, unsigned int antialiasing = 0, unsigned int major = 1, unsigned int minor = 1, unsigned int attributes = Default, bool sRgb = false)

Default constructor.

Parameters

- **depth**: Depth buffer bits
- **stencil**: Stencil buffer bits
- **antialiasing**: Antialiasing level
- **major**: Major number of the context version
- **minor**: Minor number of the context version
- **attributes**: Attribute flags of the context
- **sRgba**: sRGB capable framebuffer

Definition at line 61 of file ContextSettings.hpp.
## Member Data Documentation

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Variable Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>sf::ContextSettings::antialiasingLevel</td>
<td>Level of antialiasing. Definition at line 77 of file ContextSettings.hpp.</td>
</tr>
<tr>
<td>Uint32</td>
<td>sf::ContextSettings::attributeFlags</td>
<td>The attribute flags to create the context with. Definition at line 80 of file ContextSettings.hpp.</td>
</tr>
<tr>
<td>unsigned int</td>
<td>sf::ContextSettings::depthBits</td>
<td>Bits of the depth buffer. Definition at line 75 of file ContextSettings.hpp.</td>
</tr>
<tr>
<td>unsigned int</td>
<td>sf::ContextSettings::majorVersion</td>
<td>Major number of the context version to create. Definition at line 78 of file ContextSettings.hpp.</td>
</tr>
</tbody>
</table>
**unsigned int sf::ContextSettings::minorVersion**

Minor number of the context version to create.

Definition at line 79 of file `ContextSettings.hpp`.

**bool sf::ContextSettings::sRgbCapable**

Whether the context framebuffer is sRGB capable.

Definition at line 81 of file `ContextSettings.hpp`.

**unsigned int sf::ContextSettings::stencilBits**

Bits of the stencil buffer.

Definition at line 76 of file `ContextSettings.hpp`.

The documentation for this class was generated from the following file:
- `ContextSettings.hpp`
sf::Event Class Reference
Window module

Defines a system event and its parameters. More...

#include <Event.hpp>
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>JoystickButtonEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joystick buttons events parameters (JoystickButtonPressed, JoystickButtonReleased)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>JoystickConnectEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joystick connection events parameters (JoystickConnected, JoystickDisconnected)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>JoystickMoveEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joystick axis move event parameters (JoystickMoved)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>KeyEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard event parameters (KeyPressed, KeyReleased)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>MouseButtonEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse buttons events parameters (MouseButtonPressed, MouseButtonReleased)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>MouseMoveEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse move event parameters (MouseMoved)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>MouseWheelEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse wheel events parameters (MouseWheelMoved)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>MouseWheelScrollEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse wheel events parameters (MouseWheelScrolled)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>SensorEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor event parameters (SensorChanged)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>SizeEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size events parameters (Resized)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>TextEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>struct</td>
<td>TouchEvent</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>Touch</td>
<td>events parameters (TouchBegan, TouchMoved, TouchEnded)</td>
</tr>
</tbody>
</table>
### Public Types

<table>
<thead>
<tr>
<th>enum</th>
<th>EventType {</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Closed, Resized, LostFocus, GainedFocus,</td>
</tr>
<tr>
<td></td>
<td>TextEntered, KeyPressed, KeyReleased, MouseWheelMoved,</td>
</tr>
<tr>
<td></td>
<td>MouseWheelScrolled, MouseButtonDown, MouseButtonReleased,</td>
</tr>
<tr>
<td></td>
<td>MouseEntered, MouseLeft, JoystickButtonPressed, JoystickButtonReleased,</td>
</tr>
<tr>
<td></td>
<td>JoystickMoved, JoystickConnected, JoystickDisconnected, TouchMove,</td>
</tr>
<tr>
<td></td>
<td>TouchMoved, TouchEnded, SensorChanged, Count</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Enumeration of the different types of events. [More...](#)
## Public Attributes

<table>
<thead>
<tr>
<th>EventType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EventType</strong></td>
<td>type Type of the event. More...</td>
</tr>
<tr>
<td>union {</td>
<td></td>
</tr>
<tr>
<td>SizeEvent</td>
<td>size Size event parameters (</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>key Key event parameters (</td>
</tr>
<tr>
<td>TextEvent</td>
<td>text Text event parameters (</td>
</tr>
<tr>
<td>MouseEventArgs</td>
<td>mouseMove Mouse move event parameters (</td>
</tr>
<tr>
<td>MouseButtonEvent</td>
<td>mouseButton Mouse button event parameters (</td>
</tr>
<tr>
<td>MouseWheelEvent</td>
<td>mouseWheel Mouse wheel event parameters (</td>
</tr>
<tr>
<td>MouseWheelScrollEvent</td>
<td>mouseWheelScroll Mouse wheel event parameters (</td>
</tr>
</tbody>
</table>

More...
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>JoystickMoveEvent</td>
<td>joystickMove</td>
</tr>
<tr>
<td>JoystickButtonEvent</td>
<td>joystickButton</td>
</tr>
<tr>
<td>JoystickConnectEvent</td>
<td>joystickConnect</td>
</tr>
<tr>
<td>TouchEvent</td>
<td>touch</td>
</tr>
<tr>
<td>SensorEvent</td>
<td>sensor</td>
</tr>
</tbody>
</table>

Joystick move event parameters:

- Joystick button event parameters:
  - Event::JoystickButtonPressed
  - Event::JoystickButtonReleased

Joystick (dis)connect event parameters:

- Event::JoystickConnected
- Event::JoystickDisconnected

Touch events parameters:

- Event::TouchMoved
- Event::TouchEnded

Sensor event parameters:

- More...
Detailed Description

Defines a system event and its parameters.

sf::Event holds all the informations about a system event that just happened.

Events are retrieved using the sf::Window::pollEvent and sf::Window::waitEvent.

A sf::Event instance contains the type of the event (mouse moved, key pressed, etc.) as well as the details about this particular event. Please note that the event parameters which means that only the member matching the type of the event will be filled. All other members will have undefined values and must not be read if the type of the event does not match. For example, if you received a KeyPressed event, then you must read the event.key member, while event.MouseMove or event.text will have undefined values.

Usage example:

```cpp
sf::Event event;
while (window.pollEvent(event))
{
    // Request for closing the window
    if (event.type == sf::Event::Closed)
        window.close();

    // The escape key was pressed
    if ((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Escape))
        window.close();

    // The window was resized
    if (event.type == sf::Event::Resized)
        doSomethingWithTheNewSize(event.size.width, event.size.height);

    // etc ...
}
```

Definition at line 44 of file Event.hpp.
enum sf::Event::EventType

Enumeration of the different types of events.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>The window requested to be closed (no data)</td>
</tr>
<tr>
<td>Resized</td>
<td>The window was resized (data in event.size)</td>
</tr>
<tr>
<td>LostFocus</td>
<td>The window lost the focus (no data)</td>
</tr>
<tr>
<td>GainedFocus</td>
<td>The window gained the focus (no data)</td>
</tr>
<tr>
<td>TextEntered</td>
<td>A character was entered (data in event.text)</td>
</tr>
<tr>
<td>KeyPressed</td>
<td>A key was pressed (data in event.key)</td>
</tr>
<tr>
<td>KeyReleased</td>
<td>A key was released (data in event.key)</td>
</tr>
<tr>
<td>MouseWheelMoved</td>
<td>The mouse wheel was scrolled (data in event.mouseWheel)</td>
</tr>
<tr>
<td>MouseWheelScrolled</td>
<td></td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MouseButtonPressed</td>
<td>A mouse button was pressed (data in event.mousePosition)</td>
</tr>
<tr>
<td>MouseButtonReleased</td>
<td>A mouse button was released (data in event.mousePosition)</td>
</tr>
<tr>
<td>MouseMoved</td>
<td>The mouse cursor moved (data in event.mousePosition)</td>
</tr>
<tr>
<td>MouseEntered</td>
<td>The mouse cursor entered the area of the window (no data)</td>
</tr>
<tr>
<td>MouseLeft</td>
<td>The mouse cursor left the area of the window (no data)</td>
</tr>
<tr>
<td>JoystickButtonPressed</td>
<td>A joystick button was pressed (data in event.joystickButton)</td>
</tr>
<tr>
<td>JoystickButtonReleased</td>
<td>A joystick button was released (data in event.joystickButton)</td>
</tr>
<tr>
<td>JoystickMoved</td>
<td>The joystick moved along an axis (data in event.joystickMove)</td>
</tr>
<tr>
<td>JoystickConnected</td>
<td>A joystick was connected (data in event.joystickConnect)</td>
</tr>
<tr>
<td>JoystickDisconnected</td>
<td>A joystick was disconnected (data in event.joystickConnect)</td>
</tr>
<tr>
<td>TouchBegan</td>
<td>A touch event began (data in event.touch)</td>
</tr>
<tr>
<td>TouchMoved</td>
<td>A touch moved (data in event.touch)</td>
</tr>
<tr>
<td>TouchEnded</td>
<td></td>
</tr>
</tbody>
</table>
A touch event ended (data in event.touch)

<table>
<thead>
<tr>
<th>SensorChanged</th>
<th>A sensor value changed (data in event.sensc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Keep last – the total number of event types.</td>
</tr>
</tbody>
</table>

Definition at line **187** of file *Event.hpp*. 
## Member Data Documentation

<table>
<thead>
<tr>
<th>Event Type</th>
<th>sf::Event::name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JoystickButtonEvent</strong></td>
<td>sf::Event::joystickButton</td>
</tr>
<tr>
<td>Joystick button event parameters (Event::JoystickButtonPressed, Event::JoystickButtonReleased)</td>
<td></td>
</tr>
<tr>
<td>Definition at line 231 of file Event.hpp.</td>
<td></td>
</tr>
<tr>
<td><strong>JoystickConnectEvent</strong></td>
<td>sf::Event::joystickConnect</td>
</tr>
<tr>
<td>Joystick (dis)connect event parameters (Event::JoystickConnected, Event::JoystickDisconnected)</td>
<td></td>
</tr>
<tr>
<td>Definition at line 232 of file Event.hpp.</td>
<td></td>
</tr>
<tr>
<td><strong>JoystickMoveEvent</strong></td>
<td>sf::Event::joystickMove</td>
</tr>
<tr>
<td>Joystick move event parameters (Event::JoystickMoved)</td>
<td></td>
</tr>
<tr>
<td>Definition at line 230 of file Event.hpp.</td>
<td></td>
</tr>
<tr>
<td><strong>KeyEvent</strong></td>
<td>sf::Event::key</td>
</tr>
<tr>
<td>Key event parameters (Event::KeyPressed, Event::KeyReleased)</td>
<td></td>
</tr>
<tr>
<td>Definition at line 224 of file Event.hpp.</td>
<td></td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MouseButtonEvent sf::Event::mouseButton</td>
<td>Mouse button event parameters (Event::MouseButtonPressed, Event::MouseButtonReleased)</td>
</tr>
<tr>
<td></td>
<td>Definition at line 227 of file Event.hpp.</td>
</tr>
<tr>
<td>MouseMoveEvent sf::Event::mouseMove</td>
<td>Mouse move event parameters (Event::MouseMoved)</td>
</tr>
<tr>
<td></td>
<td>Definition at line 226 of file Event.hpp.</td>
</tr>
<tr>
<td>MouseWheelEvent sf::Event::mouseWheel</td>
<td>Mouse wheel event parameters (Event::MouseWheelMoved) (deprecated)</td>
</tr>
<tr>
<td></td>
<td>Definition at line 228 of file Event.hpp.</td>
</tr>
<tr>
<td>MouseWheelScrollEvent sf::Event::mouseWheelScroll</td>
<td>Mouse wheel event parameters (Event::MouseWheelScrolled)</td>
</tr>
<tr>
<td></td>
<td>Definition at line 229 of file Event.hpp.</td>
</tr>
<tr>
<td>SensorEvent sf::Event::sensor</td>
<td></td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Sensor event</td>
<td>Parameters (Event::SensorChanged)</td>
</tr>
<tr>
<td>SizeEvent</td>
<td>sf::Event::size</td>
</tr>
<tr>
<td>TextEvent</td>
<td>sf::Event::text</td>
</tr>
<tr>
<td>TouchEvent</td>
<td>sf::Event::touch</td>
</tr>
<tr>
<td>EventType</td>
<td>sf::Event::type</td>
</tr>
</tbody>
</table>
The documentation for this class was generated from the following file:

- Event.hpp

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila. :: Documentation generated by doxygen.
sf::Event::JoystickButtonEvent Struct Reference

Joystick buttons events parameters (JoystickButtonPressed, JoystickButtonReleased)

#include <Event.hpp>
Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>joystickId</td>
<td>Index of the joystick (in range [0 .. Joystick::Count - 1]) More...</td>
</tr>
<tr>
<td>unsigned int</td>
<td>button</td>
<td>Index of the button that has been pressed (in range [0 .. Joystick::ButtonCount])</td>
</tr>
</tbody>
</table>
Detailed Description

Joystick buttons events parameters (JoystickButtonPressed, JoystickButtonReleased).

Definition at line 154 of file Event.hpp.
Member Data Documentation

**unsigned int sf::Event::JoystickButtonEvent::button**

Index of the button that has been pressed (in range \([0 .. Joystick::Button\)]\(\)

Definition at line 157 of file Event.hpp.

**unsigned int sf::Event::JoystickButtonEvent::joystickId**

Index of the joystick (in range \([0 .. Joystick::Count - 1]\))

Definition at line 156 of file Event.hpp.

The documentation for this struct was generated from the following file:
- Event.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Event::JoystickConnectEvent Struct Reference

Joystick connection events parameters (JoystickConnected, JoystickDisconnected).

#include <Event.hpp>
## Public Attributes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>joystickId</td>
<td>Index of the joystick (in range [0 .. Joystick::Count - 1])</td>
</tr>
</tbody>
</table>
Detailed Description

Joystick connection events parameters (JoystickConnected, JoystickDisconnected).

Definition at line 133 of file Event.hpp.
**Member Data Documentation**

<table>
<thead>
<tr>
<th>unsigned int sf::Event::JoystickConnectEvent::joystickId</th>
</tr>
</thead>
</table>

Index of the joystick (in range [0 .. Joystick::Count - 1])

Definition at line 135 of file Event.hpp.

The documentation for this struct was generated from the following file:
- Event.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen
**sf::Event::JoystickMoveEvent Struct Reference**

Joystick axis move event parameters (JoystickMoved)  

```
#include <Event.hpp>
```
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>joystickId</td>
<td>Index of the joystick (in range [0 .. Joystick::Count - 1])</td>
</tr>
<tr>
<td>Joystick::Axis</td>
<td>axis</td>
<td>Axis on which the joystick moved.</td>
</tr>
<tr>
<td>float</td>
<td>position</td>
<td>New position on the axis (in range [-100 .. 100])</td>
</tr>
</tbody>
</table>
Detailed Description

Joystick axis move event parameters (JoystickMoved)

Definition at line 142 of file Event.hpp.
Member Data Documentation

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Joystick::Axis sf::Event::JoystickMoveEvent::axis</code></td>
<td>Axis on which the joystick moved.</td>
<td>145 of file Event.hpp</td>
</tr>
<tr>
<td><code>unsigned int sf::Event::JoystickMoveEvent::joystickId</code></td>
<td>Index of the joystick (in range [0 .. Joystick::Count - 1])</td>
<td>144 of file Event.hpp</td>
</tr>
<tr>
<td><code>float sf::Event::JoystickMoveEvent::position</code></td>
<td>New position on the axis (in range [-100 .. 100])</td>
<td>146 of file Event.hpp</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:
- Event.hpp

SFML is licensed under the terms and conditions of the zlib/png license. 
# sf::Event::KeyEvent Struct Reference

**Keyboard event parameters (KeyPressed, KeyReleased)**

```cpp
#include <Event.hpp>
```
### Public Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard::Key code</td>
<td>Code of the key that has been pressed. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool alt</td>
<td>Is the Alt key pressed? <a href="#">More...</a></td>
</tr>
<tr>
<td>bool control</td>
<td>Is the Control key pressed? <a href="#">More...</a></td>
</tr>
<tr>
<td>bool shift</td>
<td>Is the Shift key pressed? <a href="#">More...</a></td>
</tr>
<tr>
<td>bool system</td>
<td>Is the System key pressed? <a href="#">More...</a></td>
</tr>
</tbody>
</table>
**Detailed Description**

*Keyboard* event parameters (*KeyPressed, KeyReleased*)

Definition at line 62 of file *Event.hpp*. 
### Member Data Documentation

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>DefinitionLocation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bool sf::Event::KeyEvent::alt</strong></td>
<td>Is the Alt key pressed?</td>
<td>line 65 of Event.hpp</td>
</tr>
<tr>
<td><strong>Keyboard::Key sf::Event::KeyEvent::code</strong></td>
<td>Code of the key that has been pressed.</td>
<td>line 64 of Event.hpp</td>
</tr>
<tr>
<td><strong>bool sf::Event::KeyEvent::control</strong></td>
<td>Is the Control key pressed?</td>
<td>line 66 of Event.hpp</td>
</tr>
<tr>
<td><strong>bool sf::Event::KeyEvent::shift</strong></td>
<td>Is the Shift key pressed?</td>
<td>line 67 of Event.hpp</td>
</tr>
</tbody>
</table>
bool sf::Event::KeyEvent::system

Is the System key pressed?

Definition at line 68 of file Event.hpp.

The documentation for this struct was generated from the following file:
- Event.hpp
sf::Event::MouseButtonEvent Struct Reference

Mouse buttons events parameters (MouseButtonPressed, MouseButtonReleased)

#include <Event.hpp>
## Public Attributes

<table>
<thead>
<tr>
<th>Mouse::*Button</th>
<th>button</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code of the button that has been pressed. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X position of the mouse pointer, relative to the left of the</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y position of the mouse pointer, relative to the top of the</td>
</tr>
</tbody>
</table>
Detailed Description

Mouse buttons events parameters (MouseButtonPressed, MouseButtonR
Definition at line 95 of file Event.hpp.
## Member Data Documentation

<table>
<thead>
<tr>
<th><strong>Mouse::Button</strong> sf::Event::MouseButtonEvent::button</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code of the button that has been pressed.</td>
</tr>
<tr>
<td>Definition at line 97 of file Event.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>int sf::Event::MouseButtonEvent::x</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>X position of the mouse pointer, relative to the left of the owner window.</td>
</tr>
<tr>
<td>Definition at line 98 of file Event.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>int sf::Event::MouseButtonEvent::y</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Y position of the mouse pointer, relative to the top of the owner window.</td>
</tr>
<tr>
<td>Definition at line 99 of file Event.hpp.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:
- Event.hpp

SFML is licensed under the terms and conditions of the zlib/png license.  
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Event::MouseMoveEvent Struct Reference

Mouse move event parameters (MouseMove) More...

#include <Event.hpp>
Public Attributes

<table>
<thead>
<tr>
<th>int</th>
<th>x</th>
<th>X position of the mouse pointer, relative to the left of the owner window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>y</td>
<td>Y position of the mouse pointer, relative to the top of the owner window.</td>
</tr>
</tbody>
</table>
Detailed Description

Mouse move event parameters (MouseMoved)

Definition at line 84 of file Event.hpp.
# Member Data Documentation

<table>
<thead>
<tr>
<th><strong>int sf::Event::MouseMoveEvent::x</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>X position of the mouse pointer, relative to the left of the owner window.</td>
</tr>
<tr>
<td>Definition at line 86 of file <code>Event.hpp</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>int sf::Event::MouseMoveEvent::y</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Y position of the mouse pointer, relative to the top of the owner window.</td>
</tr>
<tr>
<td>Definition at line 87 of file <code>Event.hpp</code>.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:
- `Event.hpp`

SFML is licensed under the terms and conditions of the [zlib/png license](https:// licensing.libreoffice.org/license.html).  
Copyright © Laurent Gomila. :: Documentation generated by [Doxygen](https://www.doxygen.org/).
sf::Event::MouseWheelEvent Struct Reference

Mouse wheel events parameters (MouseWheelMoved) More...

#include <Event.hpp>
### Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>delta</td>
<td>Number of ticks the wheel has moved (positive is up, negative is down)</td>
</tr>
<tr>
<td>int</td>
<td>x</td>
<td>X position of the mouse pointer, relative to the left of the owner window</td>
</tr>
<tr>
<td>int</td>
<td>y</td>
<td>Y position of the mouse pointer, relative to the top of the owner window</td>
</tr>
</tbody>
</table>
Detailed Description

Mouse wheel events parameters (MouseWheelMoved)

**Deprecated:**

This event is deprecated and potentially inaccurate. Use `MouseWheelMoved`.

Definition at line 109 of file `Event.hpp`. 
### Member Data Documentation

<table>
<thead>
<tr>
<th>int sf::Event::MouseWheelEvent::delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of ticks the wheel has moved (positive is up, negative is down)</td>
</tr>
<tr>
<td>Definition at line 111 of file Event.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int sf::Event::MouseWheelEvent::x</th>
</tr>
</thead>
<tbody>
<tr>
<td>X position of the mouse pointer, relative to the left of the owner window.</td>
</tr>
<tr>
<td>Definition at line 112 of file Event.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int sf::Event::MouseWheelEvent::y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y position of the mouse pointer, relative to the top of the owner window.</td>
</tr>
<tr>
<td>Definition at line 113 of file Event.hpp.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:
- Event.hpp
sf::Event::MouseWheelScrollEvent Struct Reference

Mouse wheel events parameters (MouseWheelScrolled) More...

#include <Event.hpp>
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse::Wheel wheel</td>
<td>Which wheel (for mice with multiple ones)</td>
<td>More...</td>
</tr>
<tr>
<td>float</td>
<td>delta</td>
<td>Wheel offset (positive is up/left, negative is down/right). High-precision mice may use non-integral offsets. More...</td>
</tr>
<tr>
<td>int</td>
<td>x</td>
<td>X position of the mouse pointer, relative to the left of the owner window.</td>
</tr>
<tr>
<td>int</td>
<td>y</td>
<td>Y position of the mouse pointer, relative to the top of the owner window.</td>
</tr>
</tbody>
</table>
Detailed Description

Mouse wheel events parameters (MouseWheelScrolled)

Definition at line 120 of file Event.hpp.
### Member Data Documentation

#### float sf::Event::MouseWheelScrollEvent::delta

Wheel offset (positive is up/left, negative is down/right). High-precision offsets.

Definition at line 123 of file Event.hpp.

#### Mouse::Wheel sf::Event::MouseWheelScrollEvent::wheel

Which wheel (for mice with multiple ones)

Definition at line 122 of file Event.hpp.

#### int sf::Event::MouseWheelScrollEvent::x

X position of the mouse pointer, relative to the left of the owner window.

Definition at line 124 of file Event.hpp.

#### int sf::Event::MouseWheelScrollEvent::y

Y position of the mouse pointer, relative to the top of the owner window.
Definition at line 125 of file Event.hpp.

The documentation for this struct was generated from the following file:
- Event.hpp
sf::Event::SensorEvent Struct Reference

Sensor event parameters (SensorChanged) More...

#include <Event.hpp>
## Public Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor::Type</td>
<td>type</td>
<td>Type of the sensor.</td>
<td>More...</td>
</tr>
<tr>
<td>float x</td>
<td>x</td>
<td>Current value of the sensor on X axis.</td>
<td>More...</td>
</tr>
<tr>
<td>float y</td>
<td>y</td>
<td>Current value of the sensor on Y axis.</td>
<td>More...</td>
</tr>
<tr>
<td>float z</td>
<td>z</td>
<td>Current value of the sensor on Z axis.</td>
<td>More...</td>
</tr>
</tbody>
</table>
Detailed Description

Sensor event parameters (SensorChanged)

Definition at line 175 of file Event.hpp.
## Member Data Documentation

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
<th>Definition Line</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Sensor::Type sf::Event::SensorEvent::type</code></td>
<td></td>
<td>Type of the sensor.</td>
<td>177</td>
<td>Event.hpp</td>
</tr>
<tr>
<td><code>float sf::Event::SensorEvent::x</code></td>
<td></td>
<td>Current value of the sensor on X axis.</td>
<td>178</td>
<td>Event.hpp</td>
</tr>
<tr>
<td><code>float sf::Event::SensorEvent::y</code></td>
<td></td>
<td>Current value of the sensor on Y axis.</td>
<td>179</td>
<td>Event.hpp</td>
</tr>
<tr>
<td><code>float sf::Event::SensorEvent::z</code></td>
<td></td>
<td>Current value of the sensor on Z axis.</td>
<td>180</td>
<td>Event.hpp</td>
</tr>
</tbody>
</table>
The documentation for this struct was generated from the following file:

- Event.hpp

SFML is licensed under the terms and conditions of the *zlib/png*

Copyright © Laurent Gomila. :: Documentation generated by doxygen
sf::Event::SizeEvent Struct Reference

Size events parameters (Resized) More...

#include <Event.hpp>
### Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>width</td>
<td>New width, in pixels. (More...)</td>
</tr>
<tr>
<td>unsigned int</td>
<td>height</td>
<td>New height, in pixels. (More...)</td>
</tr>
</tbody>
</table>
Detailed Description

Size events parameters (Resized)

Definition at line 52 of file Event.hpp.
Member Data Documentation

**unsigned int sf::Event::SizeEvent::height**

New height, in pixels.

Definition at line 55 of file Event.hpp.

**unsigned int sf::Event::SizeEvent::width**

New width, in pixels.

Definition at line 54 of file Event.hpp.

The documentation for this struct was generated from the following file:
- Event.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Event::TextEvent Struct Reference

Text event parameters (TextEntered) More...

#include <Event.hpp>
## Public Attributes

<table>
<thead>
<tr>
<th>Uint32</th>
<th><strong>unicode</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>UTF-32 Unicode value of the character.</td>
<td>More...</td>
</tr>
</tbody>
</table>
Detailed Description

Text event parameters (TextEntered)

Definition at line 75 of file Event.hpp.
Member Data Documentation

**Uint32 sf::Event::TextEvent::unicode**

UTF-32 Unicode value of the character.

Definition at line 77 of file Event.hpp.

The documentation for this struct was generated from the following file:
- Event.hpp
sf::Event::TouchEvent Struct Reference

Touch events parameters (TouchBegan, TouchMoved, TouchEnded) More

#include <Event.hpp>
# Public Attributes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>finger</td>
<td>Index of the finger in case of multi-touch events. More...</td>
</tr>
<tr>
<td>int</td>
<td>x</td>
<td>X position of the touch, relative to the left of the owner window</td>
</tr>
<tr>
<td>int</td>
<td>y</td>
<td>Y position of the touch, relative to the top of the owner window</td>
</tr>
</tbody>
</table>
Detailed Description

*Touch* events parameters (*TouchBegan*, *TouchMoved*, *TouchEnded*)

Definition at line 164 of file *Event.hpp*.
## Member Data Documentation

<table>
<thead>
<tr>
<th><strong>unsigned int sf::Event::TouchEvent::finger</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Index of the finger in case of multi-touch events.</td>
</tr>
<tr>
<td>Definition at line 166 of file Event.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>int sf::Event::TouchEvent::x</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>X position of the touch, relative to the left of the owner window.</td>
</tr>
<tr>
<td>Definition at line 167 of file Event.hpp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>int sf::Event::TouchEvent::y</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Y position of the touch, relative to the top of the owner window.</td>
</tr>
<tr>
<td>Definition at line 168 of file Event.hpp.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:
- Event.hpp

---

SFML is licensed under the terms and conditions of the zlib/png license. Copyright © Laurent Gomila. Documentation generated by doxygen.
sf::GlResource Class Reference
Window module

Base class for classes that require an OpenGL context. More...

#include <GlResource.hpp>

Inheritance diagram for sf::GlResource:
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>TransientContextLock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAII helper class to temporarily lock an available context for use.</td>
</tr>
</tbody>
</table>
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlResource()</td>
<td>Default constructor.</td>
</tr>
<tr>
<td>~GlResource()</td>
<td>Destructor.</td>
</tr>
</tbody>
</table>
Static Protected Member Functions

<table>
<thead>
<tr>
<th>static void ensureGLContext()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty function for ABI compatibility, use acquireTransientContext instead.</td>
</tr>
</tbody>
</table>
Detailed Description

Base class for classes that require an OpenGL context.

This class is for internal use only, it must be the base of every class that requires an OpenGL context in order to work.

Definition at line 44 of file GlResource.hpp.
## Constructor & Destructor Documentation

<table>
<thead>
<tr>
<th>sf::GlResource::GlResource ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default constructor.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf::GlResource::~GlResource ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destructor.</td>
</tr>
</tbody>
</table>
Member Function Documentation

**static void sf::GlResource::ensureGlContext ( )**

Empty function for ABI compatibility, use acquireTransientContext instead.

The documentation for this class was generated from the following file:
- GlResource.hpp

SFML is licensed under the terms and conditions of the *zlib/png* license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::GlResource::TransientContextLock Class

RAII helper class to temporarily lock an available context for use. More...

#include <GlResource.hpp>

Inheritance diagram for sf::GlResource::TransientContextLock:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransientContextLock()</td>
<td>Default constructor.</td>
<td></td>
</tr>
<tr>
<td>~TransientContextLock()</td>
<td>Destructor.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

RAII helper class to temporarily lock an available context for use.

Definition at line 70 of file GlResource.hpp.
Constructor & Destructor Documentation

sf::GlResource::TransientContextLock::TransientContextLock ( )

Default constructor.

sf::GlResource::TransientContextLock::~TransientContextLock ( )

Destructor.

The documentation for this class was generated from the following file:
- GlResource.hpp

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Joystick Class Reference

Window module

Give access to the real-time state of the joysticks. More...

#include <Joystick.hpp>
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>Identification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure holding a joystick's identification. [More...]</td>
</tr>
</tbody>
</table>

### Public Types

**enum**  
```
{ Count = 8, ButtonCount = 32, AxisCount = 8 }
```
Constants related to joysticks capabilities. [More...](#)

**enum**  
```
Axis {
  X, Y, Z, R,
  U, V, PovX, PovY
}
```
Axes supported by SFML joysticks. [More...](#)
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static bool</td>
<td>isConnected (unsigned int joystick)</td>
<td>Check if a joystick is connected.  More...</td>
</tr>
<tr>
<td>static unsigned int</td>
<td>getButtonCount (unsigned int joystick)</td>
<td>Return the number of buttons supported by a joystick.</td>
</tr>
<tr>
<td>static bool</td>
<td>hasAxis (unsigned int joystick, Axis axis)</td>
<td>Check if a joystick supports a given axis.  More...</td>
</tr>
<tr>
<td>static bool</td>
<td>isButtonPressed (unsigned int joystick, unsigned int button)</td>
<td>Check if a joystick button is pressed.  More...</td>
</tr>
<tr>
<td>static float</td>
<td>getAxisPosition (unsigned int joystick, Axis axis)</td>
<td>Get the current position of a joystick axis.  More...</td>
</tr>
<tr>
<td>static Identification</td>
<td>getIdentification (unsigned int joystick)</td>
<td>Get the joystick information.  More...</td>
</tr>
<tr>
<td>static void</td>
<td>update ()</td>
<td>Update the states of all joysticks.  More...</td>
</tr>
</tbody>
</table>
Detailed Description

Give access to the real-time state of the joysticks.

sf::Joystick provides an interface to the state of the joysticks.

It only contains static functions, so it's not meant to be instantiated. Instead, each joystick is identified by an index that is passed to the functions of this class.

This class allows users to query the state of joysticks at any time and directly, without having to deal with a window and its events. Compared to the JoystickMoved, JoystickButtonPressed and JoystickButtonReleased events, sf::Joystick can retrieve the state of axes and buttons of joysticks at any time (you don't need to store and update a boolean on your side in order to know if a button is pressed or released), and you always get the real state of joysticks, even if they are moved, pressed or released when your window is out of focus and no event is triggered.

SFML supports:

- 8 joysticks (sf::Joystick::Count)
- 32 buttons per joystick (sf::Joystick::ButtonCount)
- 8 axes per joystick (sf::Joystick::AxisCount)

Unlike the keyboard or mouse, the state of joysticks is sometimes not directly available (depending on the OS), therefore an update() function must be called in order to update them. If you have a window with event handling, this is done automatically, you don't need to call it. If you have no window, or if you want to check joysticks state before creating one, you must call sf::Joystick::update explicitly.

Usage example:

```cpp
// Is joystick #0 connected?
```
bool connected = sf::Joystick::isConnected(0);

// How many buttons does joystick #0 support?
unsigned int buttons = sf::Joystick::getButtonCount(0);

// Does joystick #0 define a X axis?
bool hasX = sf::Joystick::hasAxis(0, sf::Joystick::X);

// Is button #2 pressed on joystick #0?
bool pressed = sf::Joystick::isButtonPressed(0, 2);

// What's the current position of the Y axis on joystick #0?
float position = sf::Joystick::getAxisPosition(0, sf::Joystick::Y);

See also
sf::Keyboard, sf::Mouse

Definition at line 41 of file Joystick.hpp.
## Member Enumeration Documentation

### anonymous enum

Constants related to joysticks capabilities.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Maximum number of supported joysticks.</td>
</tr>
<tr>
<td>ButtonCount</td>
<td>Maximum number of supported buttons.</td>
</tr>
<tr>
<td>AxisCount</td>
<td>Maximum number of supported axes.</td>
</tr>
</tbody>
</table>

Definition at line 49 of file Joystick.hpp.

### enum sf::Joystick::Axis

Axes supported by SFML joysticks.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>The X axis.</td>
</tr>
<tr>
<td>Y</td>
<td>The Y axis.</td>
</tr>
<tr>
<td>Z</td>
<td>The Z axis.</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------</td>
</tr>
<tr>
<td>R</td>
<td>The R axis.</td>
</tr>
<tr>
<td>U</td>
<td>The U axis.</td>
</tr>
<tr>
<td>V</td>
<td>The V axis.</td>
</tr>
<tr>
<td>PovX</td>
<td>The X axis of the point-of-view hat.</td>
</tr>
<tr>
<td>PovY</td>
<td>The Y axis of the point-of-view hat.</td>
</tr>
</tbody>
</table>

Definition at line 60 of file Joystick.hpp.
Member Function Documentation

static float sf::Joystick::getAxisPosition ( unsigned int joystick, Axis axis )

Get the current position of a joystick axis.

If the joystick is not connected, this function returns 0.

Parameters
- joystick Index of the joystick
- axis Axis to check

Returns
- Current position of the axis, in range [-100 .. 100]

static unsigned int sf::Joystick::getButtonCount ( unsigned int joystick )

Return the number of buttons supported by a joystick.

If the joystick is not connected, this function returns 0.

Parameters
- joystick Index of the joystick

Returns
- Number of buttons supported by the joystick
static Identification sf::Joystick::getIdentification (unsigned int joystick)

Get the joystick information.

Parameters
  joystick Index of the joystick

Returns
  Structure containing joystick information.

static bool sf::Joystick::hasAxis (unsigned int joystick, Axis axis)

Check if a joystick supports a given axis.

If the joystick is not connected, this function returns false.

Parameters
  joystick Index of the joystick
  axis Axis to check

Returns
  True if the joystick supports the axis, false otherwise

static bool sf::Joystick::isButtonPressed (unsigned int joystick, unsigned int button)


Check if a joystick button is pressed.

If the joystick is not connected, this function returns false.

**Parameters**
- **joystick** Index of the joystick
- **button** Button to check

**Returns**
True if the button is pressed, false otherwise

```
static bool sf::Joystick::isConnected ( unsigned int joystick )
```

Check if a joystick is connected.

**Parameters**
- **joystick** Index of the joystick to check

**Returns**
True if the joystick is connected, false otherwise

```
static void sf::Joystick::update ( )
```

Update the states of all joysticks.

This function is used internally by SFML, so you normally don't have to call it. However, you may need to call it if you have no window yet (or no window at all): in this case, joystick states are not updated automatically.

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

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Joystick::Identification Struct Reference

Structure holding a joystick's identification. More...

#include <Joystick.hpp>
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>name</td>
<td>Name of the joystick.</td>
</tr>
<tr>
<td>unsigned int</td>
<td>vendorId</td>
<td>Manufacturer identifier.</td>
</tr>
<tr>
<td>unsigned int</td>
<td>productId</td>
<td>Product identifier.</td>
</tr>
</tbody>
</table>
Detailed Description

Structure holding a joystick's identification.

Definition at line 76 of file Joystick.hpp.
Member Data Documentation

**String** sf::Joystick::Identification::name

Name of the joystick.

Definition at line 80 of file Joystick.hpp.

**unsigned int** sf::Joystick::Identification::productId

Product identifier.

Definition at line 82 of file Joystick.hpp.

**unsigned int** sf::Joystick::Identification::vendorId

Manufacturer identifier.

Definition at line 81 of file Joystick.hpp.

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

- Joystick.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Keyboard Class Reference
Window module

Give access to the real-time state of the keyboard. More...

#include <Keyboard.hpp>
Public Types

enum Key {
    Unknown = -1,
    A = 0,
    B,
    C,
    D,
    E,
    F,
    G,
    H,
    I,
    J,
    K,
    L,
    M,
    N,
    O,
    P,
    Q,
    R,
    S,
    T,
    U,
    V,
    W,
    X,
    Y,
    Z,
    Num0,
    Num1,
    Num2,
    Num3,
    Num4,
    Num5,
    Num6,
    Num7,
    Num8,
    Num9,
    Escape,
    LControl,
    LShift,
    LAlt,
    LSystem,
    RControl,
    RShift,
    RAlt,
    RSystem,
    Menu,
    LBracket,
    RBracket,
    SemiColon,
    Comma,
    Period,
    Quote,
    Slash,
    BackSlash,
    Tilde,
    Equal,
    Dash,
    Space,
    Return,
    BackSpace,
    Tab,
    PageUp,
    PageDown,
    End,
    Home,
    Insert,
    Delete,
    Add,
    Subtract,
    Multiply,
    Divide,
    Left,
    Right,
    Up,
    Down,
    Numpad0,
    Numpad1,
    Numpad2,
    Numpad3,
    Numpad4,
    Numpad5,
    Numpad6,
    Numpad7,
    Numpad8,
    Numpad9,
    F1,
    F2,
    F3,
    F4,
    F5,
    F6,
    F7,
    F8,
    F9,
    F10,
    F11,
    F12,
    F13,
    F14,
    F15,
    Pause,
    KeyCount
}

Key codes. More...
### Static Public Member Functions

<table>
<thead>
<tr>
<th>Function Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static bool isKeyPressed (Key key)</code></td>
<td>Check if a key is pressed. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>static void setVirtualKeyboardVisible (bool visible)</code></td>
<td>Show or hide the virtual keyboard. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

Give access to the real-time state of the keyboard.

sf::Keyboard provides an interface to the state of the keyboard.

It only contains static functions (a single keyboard is assumed), so it's not

This class allows users to query the keyboard state at any time and dire

Usage example:

```cpp
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
{
    // move left...
}
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
{
    // move right...
}
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
{
    // quit...
}
```

See also

sf::Joystick, sf::Mouse, sf::Touch

Definition at line 40 of file Keyboard.hpp.
# Member Enumeration Documentation

**enum sf::Keyboard::Key**

Key codes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>Unhandled key.</td>
</tr>
<tr>
<td>A</td>
<td>The A key.</td>
</tr>
<tr>
<td>B</td>
<td>The B key.</td>
</tr>
<tr>
<td>C</td>
<td>The C key.</td>
</tr>
<tr>
<td>D</td>
<td>The D key.</td>
</tr>
<tr>
<td>E</td>
<td>The E key.</td>
</tr>
<tr>
<td>F</td>
<td>The F key.</td>
</tr>
<tr>
<td>G</td>
<td>The G key.</td>
</tr>
<tr>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td>The I key.</td>
</tr>
<tr>
<td><strong>J</strong></td>
<td>The J key.</td>
</tr>
<tr>
<td><strong>K</strong></td>
<td>The K key.</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td>The L key.</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>The M key.</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>The N key.</td>
</tr>
<tr>
<td><strong>O</strong></td>
<td>The O key.</td>
</tr>
<tr>
<td><strong>P</strong></td>
<td>The P key.</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td>The Q key.</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td>The R key.</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>The S key.</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>The T key.</td>
</tr>
<tr>
<td><strong>U</strong></td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>---------------</td>
</tr>
<tr>
<td>U</td>
<td>The U key.</td>
</tr>
<tr>
<td>V</td>
<td>The V key.</td>
</tr>
<tr>
<td>W</td>
<td>The W key.</td>
</tr>
<tr>
<td>X</td>
<td>The X key.</td>
</tr>
<tr>
<td>Y</td>
<td>The Y key.</td>
</tr>
<tr>
<td>Z</td>
<td>The Z key.</td>
</tr>
<tr>
<td>Num0</td>
<td>The 0 key.</td>
</tr>
<tr>
<td>Num1</td>
<td>The 1 key.</td>
</tr>
<tr>
<td>Num2</td>
<td>The 2 key.</td>
</tr>
<tr>
<td>Num3</td>
<td>The 3 key.</td>
</tr>
<tr>
<td>Num4</td>
<td>The 4 key.</td>
</tr>
<tr>
<td>Num5</td>
<td>The 5 key.</td>
</tr>
<tr>
<td>Num6</td>
<td>The 6 key.</td>
</tr>
<tr>
<td>Num7</td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>The 7 key.</td>
</tr>
<tr>
<td>Num8</td>
<td>The 8 key.</td>
</tr>
<tr>
<td>Num9</td>
<td>The 9 key.</td>
</tr>
<tr>
<td>Escape</td>
<td>The Escape key.</td>
</tr>
<tr>
<td>LControl</td>
<td>The left Control key.</td>
</tr>
<tr>
<td>LShift</td>
<td>The left Shift key.</td>
</tr>
<tr>
<td>LAlt</td>
<td>The left Alt key.</td>
</tr>
</tbody>
</table>
| LSystem        | The left OS specific key: window (Windows and Linux), apple (MacOS X), ...
| RControl       | The right Control key.          |
| RShift         | The right Shift key.            |
| RAlt           | The right Alt key.              |
| RSystem        | The right OS specific key: window (Windows and Linux), apple (MacOS X), ...
<p>| Menu           | The Menu key.                   |
| LBracket       |                                |</p>
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[</td>
<td>The [ key.</td>
</tr>
<tr>
<td>RBracket</td>
<td>The ] key.</td>
</tr>
<tr>
<td>SemiColon</td>
<td>The ; key.</td>
</tr>
<tr>
<td>Comma</td>
<td>The , key.</td>
</tr>
<tr>
<td>Period</td>
<td>The . key.</td>
</tr>
<tr>
<td>Quote</td>
<td>The ' key.</td>
</tr>
<tr>
<td>Slash</td>
<td>The / key.</td>
</tr>
<tr>
<td>BackSlash</td>
<td>The \ key.</td>
</tr>
<tr>
<td>Tilde</td>
<td>The ~ key.</td>
</tr>
<tr>
<td>Equal</td>
<td>The = key.</td>
</tr>
<tr>
<td>Dash</td>
<td>The - key.</td>
</tr>
<tr>
<td>Space</td>
<td>The Space key.</td>
</tr>
<tr>
<td>Return</td>
<td>The Return key.</td>
</tr>
<tr>
<td>BackSpace</td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Backspace</td>
<td>The Backspace key.</td>
</tr>
<tr>
<td>Tab</td>
<td>The Tabulation key.</td>
</tr>
<tr>
<td>PageUp</td>
<td>The Page up key.</td>
</tr>
<tr>
<td>PageDown</td>
<td>The Page down key.</td>
</tr>
<tr>
<td>End</td>
<td>The End key.</td>
</tr>
<tr>
<td>Home</td>
<td>The Home key.</td>
</tr>
<tr>
<td>Insert</td>
<td>The Insert key.</td>
</tr>
<tr>
<td>Delete</td>
<td>The Delete key.</td>
</tr>
<tr>
<td>Add</td>
<td>The + key.</td>
</tr>
<tr>
<td>Subtract</td>
<td>The - key.</td>
</tr>
<tr>
<td>Multiply</td>
<td>The * key.</td>
</tr>
<tr>
<td>Divide</td>
<td>The / key.</td>
</tr>
<tr>
<td>Left</td>
<td>Left arrow.</td>
</tr>
<tr>
<td>Right</td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Right</td>
<td>Right arrow.</td>
</tr>
<tr>
<td>Up</td>
<td>Up arrow.</td>
</tr>
<tr>
<td>Down</td>
<td>Down arrow.</td>
</tr>
<tr>
<td>Numpad0</td>
<td>The numpad 0 key.</td>
</tr>
<tr>
<td>Numpad1</td>
<td>The numpad 1 key.</td>
</tr>
<tr>
<td>Numpad2</td>
<td>The numpad 2 key.</td>
</tr>
<tr>
<td>Numpad3</td>
<td>The numpad 3 key.</td>
</tr>
<tr>
<td>Numpad4</td>
<td>The numpad 4 key.</td>
</tr>
<tr>
<td>Numpad5</td>
<td>The numpad 5 key.</td>
</tr>
<tr>
<td>Numpad6</td>
<td>The numpad 6 key.</td>
</tr>
<tr>
<td>Numpad7</td>
<td>The numpad 7 key.</td>
</tr>
<tr>
<td>Numpad8</td>
<td>The numpad 8 key.</td>
</tr>
<tr>
<td>Numpad9</td>
<td>The numpad 9 key.</td>
</tr>
<tr>
<td>F1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The F1 key.</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
</tr>
<tr>
<td>F2</td>
<td>The F2 key.</td>
</tr>
<tr>
<td>F3</td>
<td>The F3 key.</td>
</tr>
<tr>
<td>F4</td>
<td>The F4 key.</td>
</tr>
<tr>
<td>F5</td>
<td>The F5 key.</td>
</tr>
<tr>
<td>F6</td>
<td>The F6 key.</td>
</tr>
<tr>
<td>F7</td>
<td>The F7 key.</td>
</tr>
<tr>
<td>F8</td>
<td>The F8 key.</td>
</tr>
<tr>
<td>F9</td>
<td>The F9 key.</td>
</tr>
<tr>
<td>F10</td>
<td>The F10 key.</td>
</tr>
<tr>
<td>F11</td>
<td>The F11 key.</td>
</tr>
<tr>
<td>F12</td>
<td>The F12 key.</td>
</tr>
<tr>
<td>F13</td>
<td>The F13 key.</td>
</tr>
<tr>
<td>F14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>F14</td>
<td>The F14 key.</td>
</tr>
<tr>
<td>F15</td>
<td>The F15 key.</td>
</tr>
<tr>
<td>Pause</td>
<td>The Pause key.</td>
</tr>
<tr>
<td>KeyCount</td>
<td>Keep last – the total number of keyboard keys.</td>
</tr>
</tbody>
</table>

Definition at line 48 of file `Keyboard.hpp`. 
Member Function Documentation

static bool sf::Keyboard::isKeyPressed ( Key key )

Check if a key is pressed.

Parameters

  key Key to check

Returns

  True if the key is pressed, false otherwise

static void sf::Keyboard::setVirtualKeyboardVisible ( bool visible )

Show or hide the virtual keyboard.

Warning: the virtual keyboard is not supported on all systems. It will typ
OSes (Android, iOS) but not on desktop OSes (Windows, Linux, ...).

If the virtual keyboard is not available, this function does nothing.

Parameters

  visible True to show, false to hide

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

- Keyboard.hpp
sf::Mouse Class Reference

Window module

Give access to the real-time state of the mouse. More...

#include <Mouse.hpp>
### Public Types

<table>
<thead>
<tr>
<th>enum</th>
<th>Button {</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Left, Right, Middle, XButton1,</td>
</tr>
<tr>
<td></td>
<td>XButton2, ButtonCount</td>
</tr>
<tr>
<td></td>
<td>} Mouse buttons. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>enum</th>
<th>Wheel { VerticalWheel, HorizontalWheel }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mouse wheels. More...</td>
</tr>
</tbody>
</table>
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Static type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static bool</td>
<td>isButtonPressed (Button button)</td>
<td>Check if a mouse button is pressed.</td>
</tr>
<tr>
<td>static Vector2i</td>
<td>getPosition ()</td>
<td>Get the current position of the mouse in desktop coordinates.</td>
</tr>
<tr>
<td>static Vector2i</td>
<td>getPosition (const Window &amp;relativeTo)</td>
<td>Get the current position of the mouse in window coordinates.</td>
</tr>
<tr>
<td>static void</td>
<td>setPosition (const Vector2i &amp;position)</td>
<td>Set the current position of the mouse in desktop coordinates.</td>
</tr>
<tr>
<td>static void</td>
<td>setPosition (const Vector2i &amp;position, const Window &amp;relativeTo)</td>
<td>Set the current position of the mouse in window coordinates.</td>
</tr>
</tbody>
</table>
Detailed Description

Give access to the real-time state of the mouse.

sf::Mouse provides an interface to the state of the mouse.

It only contains static functions (a single mouse is assumed), so it's not meant to be instantiated.

This class allows users to query the mouse state at any time and directly, without dealing with a window and its events. Compared to the MouseMoved, MouseButtonPressed events, sf::Mouse can retrieve the state of the cursor and the buttons at any time and update a boolean on your side in order to know if a button is pressed, released, or moved, pressed or released when no event is triggered.

The setPosition and getPosition functions can be used to change or retrieve the current position of the mouse pointer. There are two versions: one that operates in global coordinates (relative to the desktop) and one that operates in window coordinates (relative to a specific window).

Usage example:

```cpp
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
    // left click...
}

// get global mouse position
sf::Vector2i position = sf::Mouse::getPosition();

// set mouse position relative to a window
sf::Mouse::setPosition(sf::Vector2i(100, 200), window);
```

See also
- sf::Joystick
- sf::Keyboard
- sf::Touch
Definition at line 43 of file Mouse.hpp.
Member Enumeration Documentation

**enum sf::Mouse::Button**

Mouse buttons.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The left mouse button.</td>
</tr>
<tr>
<td>Right</td>
<td>The right mouse button.</td>
</tr>
<tr>
<td>Middle</td>
<td>The middle (wheel) mouse button.</td>
</tr>
<tr>
<td>XButton1</td>
<td>The first extra mouse button.</td>
</tr>
<tr>
<td>XButton2</td>
<td>The second extra mouse button.</td>
</tr>
<tr>
<td>ButtonCount</td>
<td>Keep last – the total number of mouse buttons.</td>
</tr>
</tbody>
</table>

Definition at line 51 of file Mouse.hpp.

**enum sf::Mouse::Wheel**
Mouse wheels.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VerticalWheel</td>
<td>The vertical mouse wheel.</td>
</tr>
<tr>
<td>HorizontalWheel</td>
<td>The horizontal mouse wheel.</td>
</tr>
</tbody>
</table>

Definition at line 66 of file Mouse.hpp.
### Member Function Documentation

#### static `Vector2i sf::Mouse::getPosition()`

Get the current position of the mouse in desktop coordinates.

This function returns the global position of the mouse cursor on the desktop.

**Returns**

Current position of the mouse

#### static `Vector2i sf::Mouse::getPosition(const Window & relativeTo)`

Get the current position of the mouse in window coordinates.

This function returns the current position of the mouse cursor, relative to the given window.

**Parameters**

- `relativeTo` Reference window

**Returns**

Current position of the mouse

#### static `bool sf::Mouse::isButtonPressed(Button button)`

Check if a mouse button is pressed.
Parameters  
  **button** Button to check

Returns  
  True if the button is pressed, false otherwise

```cpp
static void sf::Mouse::setPosition ( const Vector2i & position )
```

Set the current position of the mouse in desktop coordinates.

This function sets the global position of the mouse cursor on the desktop

Parameters  
  **position** New position of the mouse

```cpp
static void sf::Mouse::setPosition ( const Vector2i & position, const Window & relativeTo )
```

Set the current position of the mouse in window coordinates.

This function sets the current position of the mouse cursor, relative to the

Parameters  
  **position** New position of the mouse  
  **relativeTo** Reference window

The documentation for this class was generated from the following file:
- Mouse.hpp
Give access to the real-time state of the sensors. More...

#include <Sensor.hpp>
Public Types

| enum Type { Accelerometer, Gyroscope, Magnetometer, Gravity, UserAcceleration, Orientation, Count } |
| Sensor type. More... |
### Static Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static bool isAvailable (Type sensor)</code></td>
<td>Check if a sensor is available on the underlying platform.</td>
</tr>
<tr>
<td><code>static void setEnabled (Type sensor, bool enabled)</code></td>
<td>Enable or disable a sensor. More...</td>
</tr>
<tr>
<td><code>static Vector3f getValue (Type sensor)</code></td>
<td>Get the current sensor value. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Give access to the real-time state of the sensors.

sf::Sensor provides an interface to the state of the various sensors that a device provides. It only contains static functions, so it’s not meant to be instantiated.

This class allows users to query the sensors values at any time and directly, without having to deal with a window and its events. Compared to the SensorChanged event, sf::Sensor allows querying the sensor at any time (you don’t need to store and update its current value on your side).

Depending on the OS and hardware of the device (phone, tablet, ...), some sensor types may not be available. You should always check the availability of a sensor before trying to read it, with the sf::Sensor::isAvailable function.

You may wonder why some sensor types look so similar, for example, Accelerometer and Gravity/UserAcceleration. The first one is the raw measurement of the acceleration, the earth gravity and the user movement. The others are more precise separately, which is usually more useful. In fact they are not direct sensors, they are computed internally based on the raw acceleration and other sensors. This is exactly the same for Gyroscope vs Orientation.

Because sensors consume a non-negligible amount of current, they are all disabled by default. You must call sf::Sensor::setEnabled for each sensor in which you are interested.

Usage example:

```cpp
if (sf::Sensor::isAvailable(sf::Sensor::Gravity))
{
    // gravity sensor is available
}

// enable the gravity sensor
sf::Sensor::setEnabled(sf::Sensor::Gravity, true);
```
// get the current value of gravity
sf::Vector3f gravity = sf::Sensor::getValue(sf::Sensor::Gravity);

Definition at line 42 of file Sensor.hpp.
### Member Enumeration Documentation

**enum sf::Sensor::Type**

Sensor type.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>Measures the raw acceleration (m/s^2)</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>Measures the raw rotation rates (degrees/s)</td>
</tr>
<tr>
<td>Magnetometer</td>
<td>Measures the ambient magnetic field (micro-teslas)</td>
</tr>
<tr>
<td>Gravity</td>
<td>Measures the direction and intensity of gravity, independent of device acceleration (m/s^2)</td>
</tr>
<tr>
<td>UserAcceleration</td>
<td>Measures the direction and intensity of device acceleration relative to gravity (m/s^2)</td>
</tr>
<tr>
<td>Orientation</td>
<td>Measures the absolute 3D orientation (degrees)</td>
</tr>
<tr>
<td>Count</td>
<td>Keep last – the total number of sensor types.</td>
</tr>
</tbody>
</table>

Definition at line 50 of file `Sensor.hpp`. 
Member Function Documentation

static Vector3f sf::Sensor::getValue ( Type sensor )

Get the current sensor value.

**Parameters**

- *sensor* Sensor to read

**Returns**

The current sensor value

static bool sf::Sensor::isAvailable ( Type sensor )

Check if a sensor is available on the underlying platform.

**Parameters**

- *sensor* Sensor to check

**Returns**

True if the sensor is available, false otherwise

static void sf::Sensor::setEnabled ( Type sensor, bool enabled )
Enable or disable a sensor.

All sensors are disabled by default, to avoid consuming too much battery. Once enabled, it starts sending events of the corresponding type.

This function does nothing if the sensor is unavailable.

**Parameters**

- **sensor**  Sensor to enable
- **enabled** True to enable, false to disable

The documentation for this class was generated from the following file:
- Sensor.hpp
sf::Touch Class Reference
Window module

Give access to the real-time state of the touches. More...

#include <Touch.hpp>
### Static Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static bool isDown(unsigned int finger)</code></td>
<td>Check if a touch event is currently down. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>static Vector2i getPosition(unsigned int finger)</code></td>
<td>Get the current position of a touch in desktop coordinates</td>
</tr>
<tr>
<td><code>static Vector2i getPosition(unsigned int finger, const Window &amp;relativeTo)</code></td>
<td>Get the current position of a touch in window coordinates</td>
</tr>
</tbody>
</table>
### Detailed Description

Give access to the real-time state of the touches.

**sf::Touch** provides an interface to the state of the touches.

It only contains static functions, so it's not meant to be instantiated.

This class allows users to query the touches state at any time and directly retrieve the state of the touches at any time (you don't need to store and order to know if a touch is down), and you always get the real state of the touches when your window is out of focus and no event is triggered.

The getPosition function can be used to retrieve the current position of a touch. There are two versions: one that operates in global coordinates (relative to the desktop) and one that operates in window coordinates (relative to a specific window).

Touches are identified by an index (the "finger"), so that in multi-touch events, individual touches can be tracked correctly. As long as a finger touches the screen, it will keep the touch number, even if other fingers start or stop touching the screen in the meantime. As a consequence, active touch indices may not always be sequential (i.e. touch number 0 may be released while touch number 1 is still down).

Usage example:

```cpp
if (sf::Touch::isDown(0))
{
    // touch 0 is down
}

// get global position of touch 1
sf::Vector2i globalPos = sf::Touch::getPosition(1);

// get position of touch 1 relative to a window
sf::Vector2i relativePos = sf::Touch::getPosition(1, window);
```
See also
    sf::Joystick, sf::Keyboard, sf::Mouse

Definition at line 43 of file Touch.hpp.
 Member Function Documentation

\texttt{static Vector2i sf::Touch::getPosition ( unsigned int \textit{finger} )}

Get the current position of a touch in desktop coordinates.

This function returns the current touch position in global (desktop) coordinates.

Parameters

- \textit{finger} Finger index

Returns

- Current position of \textit{finger}, or undefined if it's not down

\texttt{static Vector2i sf::Touch::getPosition ( unsigned int \textit{finger},
\text{const Window &} \textit{relativeTo} )}

Get the current position of a touch in window coordinates.

This function returns the current touch position relative to the given window.

Parameters

- \textit{finger} Finger index
- \textit{relativeTo} Reference window

Returns

- Current position of \textit{finger}, or undefined if it's not down
static bool sf::Touch::isDown ( unsigned int finger )

Check if a touch event is currently down.

Parameters
   finger Finger index

Returns
   True if finger is currently touching the screen, false otherwise

The documentation for this class was generated from the following file:
- Touch.hpp
sf::VideoMode Class Reference
Window module

**VideoMode** defines a video mode (width, height, bpp) [More...]

```
#include <VideoMode.hpp>
```
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VideoMode ()</strong></td>
<td>Default constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>VideoMode (unsigned int modeWidth, unsigned int modeHeight, unsigned int modeBitsPerPixel=32)</strong></td>
<td>Construct the video mode with its attributes. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool isValid () const</strong></td>
<td>Tell whether or not the video mode is valid. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static VideoMode getDesktopMode()</td>
<td>Get the current desktop video mode. [More...]</td>
</tr>
<tr>
<td>static const std::vector &lt; VideoMode &gt; &amp; getFullscreenModes()</td>
<td>Retrieve all the video modes supported in fullscreen mode.</td>
</tr>
</tbody>
</table>
### Public Attributes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Name</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>width</td>
<td>Video mode width, in pixels.</td>
<td>More...</td>
</tr>
<tr>
<td>unsigned int</td>
<td>height</td>
<td>Video mode height, in pixels.</td>
<td>More...</td>
</tr>
<tr>
<td>unsigned int</td>
<td>bitsPerPixel</td>
<td>Video mode pixel depth, in bits per pixels.</td>
<td>More...</td>
</tr>
</tbody>
</table>
# Related Functions

(Note that these are not member functions.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool <code>operator==</code> (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td>Overload of <code>==</code> operator to compare two video modes.</td>
<td></td>
</tr>
<tr>
<td>bool <code>operator!=</code> (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td>Overload of <code>!=</code> operator to compare two video modes.</td>
<td></td>
</tr>
<tr>
<td>bool <code>operator&lt;</code> (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td>Overload of <code>&lt;</code> operator to compare video modes.</td>
<td></td>
</tr>
<tr>
<td>bool <code>operator&gt;</code> (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td>Overload of <code>&gt;</code> operator to compare video modes.</td>
<td></td>
</tr>
<tr>
<td>bool <code>operator&lt;=</code> (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td>Overload of <code>&lt;=</code> operator to compare video modes.</td>
<td></td>
</tr>
<tr>
<td>bool <code>operator&gt;=</code> (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td>Overload of <code>&gt;=</code> operator to compare video modes.</td>
<td></td>
</tr>
</tbody>
</table>
VideoMode defines a video mode (width, height, bpp)

A video mode is defined by a width and a height (in pixels) and a depth (in bits per pixel).

Video modes are used to setup windows (sf::Window) at creation time.

The main usage of video modes is for fullscreen mode: indeed you must allowed by the OS (which are defined by what the monitor and the graph window creation will just fail.

sf::VideoMode provides a static function for retrieving the list of all the system: getFullscreenModes().

A custom video mode can also be checked directly for fullscreen compatibility.

Additionally, sf::VideoMode provides a static function to get the mode getDesktopMode(). This allows to build windows with the same size or pix

Usage example:

```cpp
// Display the list of all the video modes available for fullscreen
std::vector<sf::VideoMode> modes = sf::VideoMode::getFullscreenModes();
for (std::size_t i = 0; i < modes.size(); ++i)
{
    sf::VideoMode mode = modes[i];
    std::cout << "Mode #" << i << ": "
                << mode.width << "x" << mode.height << " - "
                << mode.bitsPerPixel << " bpp" << std::endl;
}

// Create a window with the same pixel depth as the desktop
sf::VideoMode desktop = sf::VideoMode::getDesktopMode();
window.create(sf::VideoMode(1024, 768, desktop.bitsPerPixel), "SFML wi
```
Definition at line 41 of file VideoMode.hpp.
Constructor & Destructor Documentation

sf::VideoMode::VideoMode ( )

Default constructor.

This constructor initializes all members to 0.

sf::VideoMode::VideoMode ( unsigned int modeWidth,
                          unsigned int modeHeight,
                          unsigned int modeBitsPerPixel = 32 )

Construct the video mode with its attributes.

Parameters

modeWidth Width in pixels
modeHeight Height in pixels
modeBitsPerPixel Pixel depths in bits per pixel
Member Function Documentation

static VideoMode sf::VideoMode::getDesktopMode()

Get the current desktop video mode.

Returns
- Current desktop video mode

static const std::vector<VideoMode>& sf::VideoMode::getFullscreenModes()

Retrieve all the video modes supported in fullscreen mode.

When creating a fullscreen window, the video mode is restricted to be compatible with the graphics driver and monitor support. This function returns the complete list of all video modes supported in fullscreen mode. The returned array is sorted from best to worst, so that the first mode is the best mode (higher width, height and bits-per-pixel).

Returns
- Array containing all the supported fullscreen modes

bool sf::VideoMode::isValid() const

Tell whether or not the video mode is valid.

The validity of video modes is only relevant when using fullscreen windows.
can be used with no restriction.

**Returns**

True if the video mode is valid for fullscreen mode
bool operator!=( const VideoMode & left, const VideoMode & right )

Overload of != operator to compare two video modes.

Parameters

- **left** Left operand (a video mode)
- **right** Right operand (a video mode)

Returns

True if modes are different

bool operator< ( const VideoMode & left, const VideoMode & right )

Overload of < operator to compare video modes.

Parameters

- **left** Left operand (a video mode)
- **right** Right operand (a video mode)

Returns

True if left is lesser than right
bool operator<=( const VideoMode & left,  
const VideoMode & right
)

Overload of <= operator to compare video modes.

**Parameters**
- **left**  Left operand (a video mode)
- **right** Right operand (a video mode)

**Returns**
True if left is lesser or equal than right

bool operator==( const VideoMode & left,  
const VideoMode & right
)

Overload of == operator to compare two video modes.

**Parameters**
- **left**  Left operand (a video mode)
- **right** Right operand (a video mode)

**Returns**
True if modes are equal

bool operator>( const VideoMode & left,  
const VideoMode & right
)
Overload of `>` operator to compare video modes.

**Parameters**
- **left**  Left operand (a video mode)
- **right** Right operand (a video mode)

**Returns**
True if `left` is greater than `right`

```cpp
bool operator>=( const VideoMode & left,
               const VideoMode & right
 )
```

Overload of `>=` operator to compare video modes.

**Parameters**
- **left**  Left operand (a video mode)
- **right** Right operand (a video mode)

**Returns**
True if `left` is greater or equal than `right`
Member Data Documentation

**unsigned int sf::VideoMode::bitsPerPixel**

Video mode pixel depth, in bits per pixels.

Definition at line 104 of file VideoMode.hpp.

**unsigned int sf::VideoMode::height**

Video mode height, in pixels.

Definition at line 103 of file VideoMode.hpp.

**unsigned int sf::VideoMode::width**

Video mode width, in pixels.

Definition at line 102 of file VideoMode.hpp.

The documentation for this class was generated from the following file:
- VideoMode.hpp
Window that serves as a target for OpenGL rendering. More...

#include <Window.hpp>

Inheritance diagram for sf::Window:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Window()</code></td>
<td>Default constructor. More...</td>
</tr>
<tr>
<td><code>Window(VideoMode mode, const String &amp;title, const ContextSettings &amp;settings=ContextSettings())</code></td>
<td>Construct a new window. More...</td>
</tr>
<tr>
<td><code>Window(WindowHandle handle, const ContextSettings &amp;settings=ContextSettings())</code></td>
<td>Construct the window from an existing control.</td>
</tr>
<tr>
<td><code>virtual ~Window()</code></td>
<td>Destructor. More...</td>
</tr>
<tr>
<td><code>void create(VideoMode mode, const String &amp;title, const ContextSettings &amp;settings=ContextSettings())</code></td>
<td>Create (or recreate) the window. More...</td>
</tr>
<tr>
<td><code>void create(WindowHandle handle, const ContextSettings &amp;settings=ContextSettings())</code></td>
<td>Create (or recreate) the window from an existing control.</td>
</tr>
<tr>
<td><code>void close()</code></td>
<td>Close the window and destroy all the attached</td>
</tr>
<tr>
<td><code>bool isOpen() const</code></td>
<td>Tell whether or not the window is open. More...</td>
</tr>
<tr>
<td><code>const ContextSettings &amp; getSettings() const</code></td>
<td>Get the settings of the OpenGL context of the window. More...</td>
</tr>
</tbody>
</table>
### pollEvent
```cpp
bool pollEvent (Event &event)
```
Pops the event on top of the event queue, if any.

### waitEvent
```cpp
bool waitEvent (Event &event)
```
Wait for an event and return it. [More...](#)

### getPosition
```cpp
Vector2i getPosition () const
```
Get the position of the window. [More...](#)

### setPosition
```cpp
void setPosition (const Vector2i &position)
```
Change the position of the window on screen.

### getSize
```cpp
Vector2u getSize () const
```
Get the size of the rendering region of the window.

### setSize
```cpp
void setSize (const Vector2u &size)
```
Change the size of the rendering region of the window.

### setTitle
```cpp
void setTitle (const String &title)
```
Change the title of the window. [More...](#)

### setIcon
```cpp
void setIcon (unsigned int width, unsigned int height, const Uint8 *pixels)
```
Change the window’s icon. [More...](#)

### setVisible
```cpp
void setVisible (bool visible)
```
Show or hide the window. [More...](#)

### setVerticalSyncEnabled
```cpp
void setVerticalSyncEnabled (bool enabled)
```
Enable or disable vertical synchronization. [More...](#)

### setMouseCursorVisible
```cpp
void setMouseCursorVisible (bool visible)
```
Show or hide the mouse cursor. [More...](#)

### setMouseCursorGrabbed
```cpp
void setMouseCursorGrabbed (bool grabbed)
```
Grab or release the mouse cursor. [More...](#)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setKeyRepeatEnabled(bool enabled)</td>
<td>Enable or disable automatic key-repeat. <a href="#">More...</a></td>
</tr>
<tr>
<td>void setFramerateLimit(unsigned int limit)</td>
<td>Limit the framerate to a maximum fixed frequency.</td>
</tr>
<tr>
<td>void setJoystickThreshold(float threshold)</td>
<td>Change the joystick threshold. <a href="#">More...</a></td>
</tr>
<tr>
<td>bool setActive(bool active=true) const</td>
<td>Activate or deactivate the window as the current target for OpenGL rendering. <a href="#">More...</a></td>
</tr>
<tr>
<td>void requestFocus()</td>
<td>Request the current window to be made the active foreground window.</td>
</tr>
<tr>
<td>bool hasFocus() const</td>
<td>Check whether the window has the input focus.</td>
</tr>
<tr>
<td>void display()</td>
<td>Display on screen what has been rendered to the window so far.</td>
</tr>
<tr>
<td>WindowHandle getSystemHandle() const</td>
<td>Get the OS-specific handle of the window. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual void <strong>onCreate</strong> ()</td>
<td>Function called after the window has been created. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual void <strong>onResize</strong> ()</td>
<td>Function called after the window has been resized. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Static Private Member Functions

```cpp
static void ensureGLContext ()
Empty function for ABI compatibility, use acquireTransientCor
```
Detailed Description

**Window** that serves as a target for OpenGL rendering.

**sf::Window** is the main class of the **Window** module.

It defines an OS window that is able to receive an OpenGL rendering.

A **sf::Window** can create its own new window, or be embedded into an existing window using the `create(handle)` function. This can be useful for embedding an OpenGL rendering area into a bigger GUI with existing windows, controls, etc. It can also serve as an OpenGL rendering area into a window created by another (probably richer) GUI library.

The **sf::Window** class provides a simple interface for manipulating the window: move, resize, show/hide, control mouse cursor, etc. It also provides event handling through its `pollEvent()` function.

Note that OpenGL experts can pass their own parameters (antialiasing level, depth and stencil buffers, etc.) to the OpenGL context attached to the window, with the `sf::ContextSettings` passed as an optional argument when creating the window.

Usage example:

```cpp
// Declare and create a new window
sf::Window window(sf::VideoMode(800, 600), "SFML window");

// Limit the framerate to 60 frames per second (this step is optional)
window.setFramerateLimit(60);

// The main loop - ends as soon as the window is closed
while (window.isOpen())
{
    // Event processing
    sf::Event event;
    while (window.pollEvent(event))
    {
        // Request for closing the window
        if (event.type == sf::Event::Closed)
            break;
    }
    // Render to the window
    // Update the window
    // 
    // Display the window
    // 
    // 
    // Repeat
}
```
window.close();

// Activate the window for OpenGL rendering
window.setActive();

// OpenGL drawing commands go here...

// End the current frame and display its contents on screen
window.display();

Definition at line 57 of file Window/Window.hpp.
Constructor & Destructor Documentation

sf::Window::Window ( )

Default constructor.
This constructor doesn't actually create the window, use the other constructor.

sf::Window::Window ( VideoMode mode,
                      const String & title,
                      Uint32 style = Style::Default,
                      const ContextSettings & settings = ContextSettings )

Construct a new window.
This constructor creates the window with the size and pixel depth defined be passed to customize the look and behavior of the window (borders, title).
If style contains Style::Fullscreen, then mode must be a valid video mode.
The fourth parameter is an optional structure specifying advanced OpenGL antialiasing, depth-buffer bits, etc.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Video mode to use (defines the width, height and depth of window)</td>
</tr>
<tr>
<td>title</td>
<td>Title of the window</td>
</tr>
<tr>
<td>style</td>
<td>Window style, a bitwise OR combination of sf::Style enumerators</td>
</tr>
</tbody>
</table>
sf::Window::Window (WindowHandle handle, const ContextSettings & settings = ContextSettings())

Construct the window from an existing control.

Use this constructor if you want to create an OpenGL rendering area into an existing control.

The second parameter is an optional structure specifying advanced OpenGL context settings such as antialiasing, depth-buffer bits, etc.

Parameters

- **handle** Platform-specific handle of the control
- **settings** Additional settings for the underlying OpenGL context

virtual sf::Window::~Window ()

Destructor.

Closes the window and frees all the resources attached to it.
void sf::Window::close ()

Close the window and destroy all the attached resources.

After calling this function, the sf::Window instance remains valid and you can call window::pollEvent() or window::display() will still work (i.e. you don’t have to test window::isOpen() every time), and will have no effect on closed windows.

void sf::Window::create ( VideoMode mode, const String & title, Uint32 style = Style::Default, const ContextSettings & settings = ContextSettings )

Create (or recreate) the window.

If the window was already created, it closes it first. If style contains Style::Fullscreen, then a valid video mode.

The fourth parameter is an optional structure specifying advanced OpenGL context settings such as antialiasing, depth-buffer bits, etc.

Parameters

- mode: Video mode to use (defines the width, height and depth of window)
- title: Title of the window
**style**  Window style, a bitwise OR combination of sf::Style enumerators

**settings** Additional settings for the underlying OpenGL context

```cpp
void sf::Window::create (WindowHandle handle,
                          const ContextSettings & settings = ContextSettings())
```

Create (or recreate) the window from an existing control.

Use this function if you want to create an OpenGL rendering area into a window that was already created, it will close it first.

The second parameter is an optional structure specifying advanced OpenGL context settings such as antialiasing, depth-buffer bits, etc.

**Parameters**

- **handle**  Platform-specific handle of the control
- **settings** Additional settings for the underlying OpenGL context

```cpp
void sf::Window::display ( )
```

Display on screen what has been rendered to the window so far.

This function is typically called after all OpenGL rendering has been done to show it on screen.

```cpp
Vector2i sf::Window::getPosition ( ) const
```

Get the position of the window.
Returns
Position of the window, in pixels

See also
setPosition

const ContextSettings & sf::Window::getSettings () const

Get the settings of the OpenGL context of the window.

Note that these settings may be different from what was passed to function, if one or more settings were not supported. In this case, SFML (...

Returns
Structure containing the OpenGL context settings

Vector2u sf::Window::getSize () const

Get the size of the rendering region of the window.

The size doesn't include the titlebar and borders of the window.

Returns
Size in pixels

See also
setSize

WindowHandle sf::Window::getSystemHandle () const
Get the OS-specific handle of the window.

The type of the returned handle is sf::WindowHandle, which is a type defined by the OS. 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

**Returns**

System handle of the window

---

**bool sf::Window::hasFocus() const**

Check whether the window has the input focus.

At any given time, only one window may have the input focus to receive or most mouse events.

**Returns**

True if window has focus, false otherwise

**See also**

requestFocus

---

**bool sf::Window::isOpen() const**

Tell whether or not the window is open.

This function returns whether or not the window exists. Note that a hidden window is open (therefore this function would return true).

**Returns**

True if the window is open, false if it has been closed
virtual void sf::Window::onCreate ()

Function called after the window has been created.

This function is called so that derived classes can perform their own specific initialization as soon as the window is created.

Reimplemented in sf::RenderWindow.

virtual void sf::Window::onResize ()

Function called after the window has been resized.

This function is called so that derived classes can perform custom actions when the size of the window changes.

Reimplemented in sf::RenderWindow.

bool sf::Window::pollEvent ( Event & event )

Pop the event on top of the event queue, if any, and return it.

This function is not blocking: if there's no pending event then it will return false and leave unmodified. Note that more than one event may be present in the event queue, thus you should always call this function in a loop to make sure that you process every pending event.

```
sf::Event event;
while (window.pollEvent(event))
{
  // process event...
}
```
Parameters

    event Event to be returned

Returns

    True if an event was returned, or false if the event queue was empty

See also

    waitEvent

### void sf::Window::requestFocus ( )

Request the current window to be made the active foreground window.

At any given time, only one window may have the input focus to receive or mouse events. If a window requests focus, it only hints to the operating system that it would like to be focused. The operating system is free to deny the request. This is not to

See also

    hasFocus

### bool sf::Window::setActive ( bool active = true ) const

Activate or deactivate the window as the current target for OpenGL rendering.

A window is active only on the current thread, if you want to make it active on another thread first if it was active. Only one window can be active at a time, thus the window previously active (if any) automatically gets deactivated with requestFocus().

Parameters
**active** True to activate, false to deactivate

**Returns**
True if operation was successful, false otherwise

```cpp
void sf::Window::setFramerateLimit ( unsigned int  limit )
```

Limit the framerate to a maximum fixed frequency.

If a limit is set, the window will use a small delay after each call to `display` if frame lasted long enough to match the framerate limit. SFML will try to respect it can, but since it internally uses `sf::sleep`, whose precision depends on the underlying OS, the results may be a little unprecise as well (for example, you can get 65 FPS when

**Parameters**

- **limit** Framerate limit, in frames per seconds (use 0 to disable limit)

```cpp
void sf::Window::setIcon ( unsigned int  width, unsigned int  height, const Uint8 *  pixels )
```

Change the window's icon.

*pixels* must be an array of *width* x *height* pixels in 32-bits RGBA format.

The OS default icon is used by default.

**Parameters**

- **width** Icon's width, in pixels
- **height** Icon's height, in pixels
**pixels** Pointer to the array of pixels in memory. The pixels are copied, so you need not keep the source alive after calling this function.

See also

setTitle

---

```cpp
void sf::Window::setJoystickThreshold(float threshold)
```

Change the joystick threshold.

The joystick threshold is the value below which no JoystickMoved event will be generated.

The threshold value is 0.1 by default.

**Parameters**

- `threshold` New threshold, in the range [0, 100]

---

```cpp
void sf::Window::setKeyRepeatEnabled(bool enabled)
```

Enable or disable automatic key-repeat.

If key repeat is enabled, you will receive repeated KeyPressed events. If it is disabled, you will only get a single event when the key is pressed.

Key repeat is enabled by default.

**Parameters**

- `enabled` True to enable, false to disable

---

```cpp
void sf::Window::setMouseCursorGrabbed(bool grabbed)
```
Grab or release the mouse cursor.

If set, grabs the mouse cursor inside this window’s client area so it may bounds. Note that grabbing is only active while the window has focus.

**Parameters**

- **grabbed** True to enable, false to disable

---

```cpp
void sf::Window::setMouseCursorVisible ( bool visible )
```

Show or hide the mouse cursor.

The mouse cursor is visible by default.

**Parameters**

- **visible** True to show the mouse cursor, false to hide it

---

```cpp
void sf::Window::setPosition ( const Vector2i & position )
```

Change the position of the window on screen.

This function only works for top-level windows (i.e. it will be ignored for a child window/control).

**Parameters**

- **position** New position, in pixels

**See also**

- getPosition
### void sf::Window::setSize(const Vector2u & size)

Change the size of the rendering region of the window.

**Parameters**

- **size** New size, in pixels

**See also**

- getSize

### void sf::Window::setTitle(const String & title)

Change the title of the window.

**Parameters**

- **title** New title

**See also**

- setIcon

### void sf::Window::setVerticalSyncEnabled(bool enabled)

Enable or disable vertical synchronization.

Activating vertical synchronization will limit the number of frames disp monitor. This can avoid some visual artifacts, and limit the framerate tc across different computers).

Vertical synchronization is disabled by default.

**Parameters**

- **enabled**
**enabled** True to enable v-sync, false to deactivate it

<table>
<thead>
<tr>
<th>void sf::Window::setVisible ( bool visible )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show or hide the window.</td>
</tr>
<tr>
<td>The window is shown by default.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td><strong>visible</strong> True to show the window, false to hide it</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bool sf::Window::waitEvent ( Event &amp; event )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait for an event and return it.</td>
</tr>
<tr>
<td>This function is blocking: if there's no pending event then it will wait until function returns (and no error occurred), the event object is always valid.</td>
</tr>
<tr>
<td>This function is typically used when you have a thread that is dedicated to events handling; you want this thread to sleep as long as no new event is received.</td>
</tr>
<tr>
<td>```cpp</td>
</tr>
<tr>
<td>sf::Event event;</td>
</tr>
<tr>
<td>if (window.waitEvent(event))</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>// process event...</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td><strong>event</strong> Event to be returned</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td>False if any error occurred</td>
</tr>
</tbody>
</table>
See also

pollEvent

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

- Window/Window.hpp

SFML is licensed under the terms and conditions of the zlib/png license. Copyright © Laurent Gomila. :: Documentation generated by doxygen.
Namespace List

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

- sf
  - Glsl  Namespace with GLSL types
Here is a list of all documented namespace members with links to the namespaces they belong to:

- Bvec2 : sf::Glsl
- Bvec3 : sf::Glsl
- Bvec4 : sf::Glsl
- Ivec2 : sf::Glsl
- Ivec3 : sf::Glsl
- Ivec4 : sf::Glsl
- Mat3 : sf::Glsl
- Mat4 : sf::Glsl
- Vec2 : sf::Glsl
- Vec3 : sf::Glsl
- Vec4 : sf::Glsl
<table>
<thead>
<tr>
<th>Bvec2 : sf::Glsl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bvec3 : sf::Glsl</td>
</tr>
<tr>
<td>Bvec4 : sf::Glsl</td>
</tr>
<tr>
<td>Ivec2 : sf::Glsl</td>
</tr>
<tr>
<td>Ivec3 : sf::Glsl</td>
</tr>
<tr>
<td>Ivec4 : sf::Glsl</td>
</tr>
<tr>
<td>Mat3 : sf::Glsl</td>
</tr>
<tr>
<td>Mat4 : sf::Glsl</td>
</tr>
<tr>
<td>Vec2 : sf::Glsl</td>
</tr>
<tr>
<td>Vec3 : sf::Glsl</td>
</tr>
<tr>
<td>Vec4 : sf::Glsl</td>
</tr>
</tbody>
</table>
Here are the classes, structs, unions and interfaces with brief descriptions:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlResource</td>
<td>Base class for classes that require an OpenAL context</td>
</tr>
<tr>
<td>BlendMode</td>
<td>Blending modes for drawing</td>
</tr>
<tr>
<td>CircleShape</td>
<td>Specialized shape representing a circle</td>
</tr>
<tr>
<td>Clock</td>
<td>Utility class that measures the elapsed time</td>
</tr>
<tr>
<td>Color</td>
<td>Utility class for manipulating RGBA colors</td>
</tr>
<tr>
<td>Context</td>
<td>Class holding a valid drawing context</td>
</tr>
<tr>
<td>ContextSettings</td>
<td>Structure defining the settings of the OpenGL context window</td>
</tr>
<tr>
<td>ConvexShape</td>
<td>Specialized shape representing a convex polygon</td>
</tr>
<tr>
<td>Drawable</td>
<td>Abstract base class for objects that can be drawn</td>
</tr>
<tr>
<td>Event</td>
<td>Defines a system event and its parameters</td>
</tr>
<tr>
<td>JoystickButtonEvent</td>
<td>Joystick buttons events parameters (JoystickButtonPressed, JoystickButtonReleased)</td>
</tr>
<tr>
<td>JoystickConnectEvent</td>
<td>Joystick connection events parameters</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JoystickMoveEvent</td>
<td>Joystick axis move event parameters (JoystickMoved)</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>Keyboard event parameters (KeyPressed, KeyReleased)</td>
</tr>
<tr>
<td>MouseButtonEvent</td>
<td>Mouse buttons events parameters (MouseButtonPressed, MouseButtonReleased)</td>
</tr>
<tr>
<td>MouseMoveEvent</td>
<td>Mouse move event parameters (MouseMoved)</td>
</tr>
<tr>
<td>MouseWheelEvent</td>
<td>Mouse wheel events parameters (MouseWheelMoved)</td>
</tr>
<tr>
<td>MouseWheelScrollEvent</td>
<td>Mouse wheel events parameters (MouseWheelScrolled)</td>
</tr>
<tr>
<td>SensorEvent</td>
<td>Sensor event parameters (SensorChanged)</td>
</tr>
<tr>
<td>SizeEvent</td>
<td>Size events parameters (Resized)</td>
</tr>
<tr>
<td>TextEvent</td>
<td>Text event parameters (TextEntered)</td>
</tr>
<tr>
<td>TouchEvent</td>
<td>Touch events parameters (TouchBegan, TouchMoved, TouchEnded)</td>
</tr>
<tr>
<td>FileInputStream</td>
<td>Implementation of input stream based on file</td>
</tr>
<tr>
<td>Font</td>
<td>Class for loading and manipulating character fonts</td>
</tr>
<tr>
<td>Info</td>
<td>Holds various information about a font</td>
</tr>
<tr>
<td>Ftp</td>
<td>A FTP client</td>
</tr>
<tr>
<td>DirectoryResponse</td>
<td>Specialization of FTP response returning a directory</td>
</tr>
<tr>
<td>ListingResponse</td>
<td>Specialization of FTP response returning a filename listing</td>
</tr>
<tr>
<td>Response</td>
<td>Define a FTP response</td>
</tr>
<tr>
<td>GIResource</td>
<td>Base class for classes that require an OpenGL context</td>
</tr>
<tr>
<td>TransientContextLock</td>
<td>RAII helper class to temporarily lock an OpenGL context</td>
</tr>
<tr>
<td>Glyph</td>
<td>Structure describing a glyph</td>
</tr>
<tr>
<td>Http</td>
<td>A HTTP client</td>
</tr>
<tr>
<td>Request</td>
<td>Define a HTTP request</td>
</tr>
<tr>
<td>Response</td>
<td>Define a HTTP response</td>
</tr>
<tr>
<td>Image</td>
<td>Class for loading, manipulating and saving images</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InputSoundFile</td>
<td>Provide read access to sound files</td>
</tr>
<tr>
<td>InputStream</td>
<td>Abstract class for custom file input streams</td>
</tr>
<tr>
<td>IpAddress</td>
<td>Encapsulate an IPv4 network address</td>
</tr>
<tr>
<td>Joystick</td>
<td>Give access to the real-time state of the joystick</td>
</tr>
<tr>
<td>Identification</td>
<td>Structure holding a joystick’s identification</td>
</tr>
<tr>
<td>Keyboard</td>
<td>Give access to the real-time state of the keyboard</td>
</tr>
<tr>
<td>Listener</td>
<td>The audio listener is the point in the scene from where all the sounds are heard</td>
</tr>
<tr>
<td>Lock</td>
<td>Automatic wrapper for locking and unlocking mutexes</td>
</tr>
<tr>
<td>MemoryInputOutputStream</td>
<td>Implementation of input stream based on a memory chunk</td>
</tr>
<tr>
<td>Mouse</td>
<td>Give access to the real-time state of the mouse</td>
</tr>
<tr>
<td>Music</td>
<td>Streamed music played from an audio file</td>
</tr>
<tr>
<td>Mutex</td>
<td>Blocks concurrent access to shared resources</td>
</tr>
<tr>
<td>NonCopyable</td>
<td>Utility class that makes any derived class non-copyable</td>
</tr>
<tr>
<td>OutputSoundFile</td>
<td>Provide write access to sound files</td>
</tr>
<tr>
<td>Packet</td>
<td>Utility class to build blocks of data to transfer</td>
</tr>
<tr>
<td>Rect</td>
<td>Utility class for manipulating 2D axis aligned rectangles</td>
</tr>
<tr>
<td>RectangleShape</td>
<td>Specialized shape representing a rectangle</td>
</tr>
<tr>
<td>RenderStates</td>
<td>Define the states used for drawing to a render target</td>
</tr>
<tr>
<td>RenderTarget</td>
<td>Base class for all render targets (window, texture, ...)</td>
</tr>
<tr>
<td>RenderTexture</td>
<td>Target for off-screen 2D rendering into a texture</td>
</tr>
<tr>
<td>RenderWindow</td>
<td>Window that can serve as a target for 2D drawing</td>
</tr>
<tr>
<td>Sensor</td>
<td>Give access to the real-time state of the sensors</td>
</tr>
<tr>
<td>Shader</td>
<td>Shader class (vertex, geometry and fragment)</td>
</tr>
<tr>
<td>CurrentTextureType</td>
<td>Special type that can be passed to setUniform() to texture of the object being drawn</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Shape</td>
<td>Base class for textured shapes with outline</td>
</tr>
<tr>
<td>Socket</td>
<td>Base class for all the socket types</td>
</tr>
<tr>
<td>SocketSelector</td>
<td>Multiplexer that allows to read from multiple sockets</td>
</tr>
<tr>
<td>Sound</td>
<td>Regular sound that can be played in the audio environment</td>
</tr>
<tr>
<td>SoundBuffer</td>
<td>Storage for audio samples defining a sound</td>
</tr>
<tr>
<td>SoundBufferRecorder</td>
<td>Specialized SoundRecorder which stores the captured audio data into a sound buffer</td>
</tr>
<tr>
<td>SoundFileFactory</td>
<td>Manages and instantiates sound file readers</td>
</tr>
<tr>
<td>SoundFileReader</td>
<td>Abstract base class for sound file decoding</td>
</tr>
<tr>
<td>Info</td>
<td>Structure holding the audio properties of a sound file</td>
</tr>
<tr>
<td>SoundFileWriter</td>
<td>Abstract base class for sound file encoding</td>
</tr>
<tr>
<td>SoundRecorder</td>
<td>Abstract base class for capturing sound data</td>
</tr>
<tr>
<td>SoundSource</td>
<td>Base class defining a sound's properties</td>
</tr>
<tr>
<td>SoundStream</td>
<td>Abstract base class for streamed audio</td>
</tr>
<tr>
<td>Chunk</td>
<td>Structure defining a chunk of audio data</td>
</tr>
<tr>
<td>Sprite</td>
<td>Drawable representation of a texture, with its own transformations, color, etc</td>
</tr>
<tr>
<td>String</td>
<td>Utility string class that automatically handles conversions and encodings</td>
</tr>
<tr>
<td>TcpListener</td>
<td>Socket that listens to new TCP connections</td>
</tr>
<tr>
<td>TcpSocket</td>
<td>Specialized socket using the TCP protocol</td>
</tr>
<tr>
<td>Text</td>
<td>Graphical text that can be drawn to a render target</td>
</tr>
<tr>
<td>Texture</td>
<td>Image living on the graphics card that can be used for drawing</td>
</tr>
<tr>
<td>Thread</td>
<td>Utility class to manipulate threads</td>
</tr>
<tr>
<td>ThreadLocal</td>
<td>Defines variables with thread-local storage</td>
</tr>
<tr>
<td>ThreadLocalPtr</td>
<td>Pointer to a thread-local variable</td>
</tr>
<tr>
<td>Time</td>
<td>Represents a time value</td>
</tr>
<tr>
<td>Identifier</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Touch</td>
<td>Give access to the real-time state of the touches</td>
</tr>
<tr>
<td>Transform</td>
<td>Define a 3x3 transform matrix</td>
</tr>
<tr>
<td>Transformable</td>
<td>Decomposed transform defined by a position, rotation, and scale</td>
</tr>
<tr>
<td>UdpSocket</td>
<td>Specialized socket using the UDP protocol</td>
</tr>
<tr>
<td>Utf</td>
<td>Utility class providing generic functions</td>
</tr>
<tr>
<td>Utf&lt; 16 &gt;</td>
<td>Specialization of the Utf template for UTF-16</td>
</tr>
<tr>
<td>Utf&lt; 32 &gt;</td>
<td>Specialization of the Utf template for UTF-32</td>
</tr>
<tr>
<td>Utf&lt; 8 &gt;</td>
<td>Specialization of the Utf template for UTF-8</td>
</tr>
<tr>
<td>Vector2</td>
<td>Utility template class for manipulating 2D vectors</td>
</tr>
<tr>
<td>Vector3</td>
<td>Utility template class for manipulating 3D vectors</td>
</tr>
<tr>
<td>Vertex</td>
<td>Define a point with color and texture coordinates</td>
</tr>
<tr>
<td>VertexArray</td>
<td>Define a set of one or more 2D primitives</td>
</tr>
<tr>
<td>VideoMode</td>
<td>VideoMode defines a video mode (width, height, bpp)</td>
</tr>
<tr>
<td>View</td>
<td>2D camera that defines what region is shown on the screen</td>
</tr>
<tr>
<td>Window</td>
<td>Window that serves as a target for OpenGL rendering</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Utf< 16 > Class Template Reference

Specialization of the Utf template for UTF-16. More...

#include <Utf.hpp>
### Static Public Member Functions

**template<typename In >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>decode</code> (In begin, In end, Uint32 &amp;output, Uint32 replacement=0)</td>
<td>Decode a single UTF-16 character. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>encode</code> (Uint32 input, Out output, Uint16 replacement=0)</td>
<td>Encode a single UTF-16 character. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>next</code> (In begin, In end)</td>
<td>Advance to the next UTF-16 character. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count</code> (In begin, In end)</td>
<td>Count the number of characters of a UTF-16 sequence</td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fromAnsi</code> (In begin, In end, Out output, const std::locale &amp;locale=std::locale())</td>
<td>Convert an ANSI characters range to UTF-16. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fromWide</code> (In begin, In end, Out output)</td>
<td>Convert a wide characters range to UTF-16. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fromLatin1</code> (In begin, In end, Out output)</td>
<td>Convert a latin-1 (ISO-5589-1) characters range to UTF-16</td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toAnsi</code> (In begin, In end, Out output, char replacement=0, &amp;locale=std::locale())</td>
<td>Convert an UTF-16 characters range to ANSI characters</td>
</tr>
</tbody>
</table>
template<typename In, typename Out >
  static Out  toWide (In begin, In end, Out output, wchar_t replacement=0)
  Convert an UTF-16 characters range to wide characters

template<typename In, typename Out >
  static Out  toLatin1 (In begin, In end, Out output, char replacement=0)
  Convert an UTF-16 characters range to latin-1 (ISO-5589-1)

template<typename In, typename Out >
  static Out  toUtf8 (In begin, In end, Out output)
  Convert a UTF-16 characters range to UTF-8. More...

template<typename In, typename Out >
  static Out  toUtf16 (In begin, In end, Out output)
  Convert a UTF-16 characters range to UTF-16. More...

template<typename In, typename Out >
  static Out  toUtf32 (In begin, In end, Out output)
  Convert a UTF-16 characters range to UTF-32. More...
Detailed Description

template<>  
class sf::Utf< 16 >

Specialization of the Utf template for UTF-16.

Definition at line 255 of file Utf.hpp.
Member Function Documentation

template<typename In >
static std::size_t sf::Utf< 16 >::count ( In begin,
                   In end )

Count the number of characters of a UTF-16 sequence.
This function is necessary for multi-elements encodings, as a single character may use more than 1 storage element, thus the total size can be different from (begin - end).

Parameters

begin Iterator pointing to the beginning of the input sequence
end Iterator pointing to the end of the input sequence

Returns

Iterator pointing to one past the last read element of the input sequence

template<typename In >
static In sf::Utf< 16 >::decode ( In begin,
                   In end,
                   Uint32 & output,
                   Uint32 replacement = 0 )

Decode a single UTF-16 character.
Decoding a character means finding its unique 32-bits code (called the codepoint) in the Unicode standard.

**Parameters**
- `begin` Iterator pointing to the beginning of the input sequence
- `end` Iterator pointing to the end of the input sequence
- `output` Codepoint of the decoded UTF-16 character
- `replacement` Replacement character to use in case the UTF-8 sequence is invalid

**Returns**
- Iterator pointing to one past the last read element of the input sequence

```cpp
template<typename Out>
static Out sf::Utf< 16 >::encode ( Uint32 input, Out output, Uint16 replacement = 0 )
```

Encode a single UTF-16 character.

Encoding a character means converting a unique 32-bits code (called the codepoint) in the target encoding, UTF-16.

**Parameters**
- `input` Codepoint to encode as UTF-16
- `output` Iterator pointing to the beginning of the output sequence
- `replacement` Replacement for characters not convertible to UTF-16

**Returns**
- Iterator to the end of the output sequence which has been written
template<typename In , typename Out >

static Out sf::Utf< 16 >::fromAnsi ( In begin, In end, Out output, const std::locale & locale = std::locale() )

Convert an ANSI characters range to UTF-16.

The current global locale will be used by default, unless you pass a custom.

Parameters

- begin  Iterator pointing to the beginning of the input sequence
- end  Iterator pointing to the end of the input sequence
- output Iterator pointing to the beginning of the output sequence
- locale  Locale to use for conversion

Returns

Iterator to the end of the output sequence which has been written.

template<typename In , typename Out >

static Out sf::Utf< 16 >::fromLatin1 ( In begin, In end, Out output )

Convert a latin-1 (ISO-5589-1) characters range to UTF-16.

Parameters

- begin  Iterator pointing to the beginning of the input sequence
- end  Iterator pointing to the end of the input sequence
- output Iterator pointing to the beginning of the output sequence
Returns
Iterator to the end of the output sequence which has been written

```
template<typename In , typename Out >
static Out sf::Utf< 16 >::fromWide ( In begin, In end, Out output )
```

Convert a wide characters range to UTF-16.

Parameters
- **begin**  Iterator pointing to the beginning of the input sequence
- **end**  Iterator pointing to the end of the input sequence
- **output**  Iterator pointing to the beginning of the output sequence

Returns
Iterator to the end of the output sequence which has been written

```
template<typename In >
static In sf::Utf< 16 >::next ( In begin, In end )
```

Advance to the next UTF-16 character.

This function is necessary for multi-elements encodings, as a single character may use more than 1 storage element.

Parameters
begin Iterator pointing to the beginning of the input sequence
end Iterator pointing to the end of the input sequence

Returns
Iterator pointing to one past the last read element of the input sequence

template<typename In , typename Out >
static Out sf::Utf< 16 >::toAnsi ( In begin, In end, Out output, char replacement = 0 const std::locale & locale = std::locale() )

Convert an UTF-16 characters range to ANSI characters.
The current global locale will be used by default, unless you pass a custom

Parameters

begin Iterator pointing to the beginning of the input sequence
end Iterator pointing to the end of the input sequence
output Iterator pointing to the beginning of the output sequence
replacement Replacement for characters not convertible to ANSI (use 0 to skip them)
locale Locale to use for conversion

Returns
Iterator to the end of the output sequence which has been written

template<typename In , typename Out >
static Out sf::Utf< 16 >::toLatin1 ( In begin, In end, Out output, char replacement = 0 const std::locale & locale = std::locale() )
Convert an UTF-16 characters range to latin-1 (ISO-5589-1) characters.

Parameters

- **begin**: Iterator pointing to the beginning of the input sequence
- **end**: Iterator pointing to the end of the input sequence
- **output**: Iterator pointing to the beginning of the output sequence
- **replacement**: Replacement for characters not convertible to wide (use 0 to skip them)

Returns

Iterator to the end of the output sequence which has been written

```
template<typename In , typename Out >
static Out sf::Utf< 16 >::toUtf16 ( In begin, In end, Out output )
```

Convert a UTF-16 characters range to UTF-16.

This functions does nothing more than a direct copy; it is defined only to allow other specializations of the sf::Utf<> template, and allow generic code to

Parameters

- **begin**: Iterator pointing to the beginning of the input sequence
- **end**: Iterator pointing to the end of the input sequence
- **output**: Iterator pointing to the beginning of the output sequence

Returns
template<typename In , typename Out >

static Out sf::Utf<16>::toUtf32 ( In begin, In end, Out output )

Convert a UTF-16 characters range to UTF-32.

**Parameters**
- **begin** Iterator pointing to the beginning of the input sequence
- **end** Iterator pointing to the end of the input sequence
- **output** Iterator pointing to the beginning of the output sequence

**Returns**
Iterator to the end of the output sequence which has been written

---

template<typename In , typename Out >

static Out sf::Utf<16>::toUtf8 ( In begin, In end, Out output )

Convert a UTF-16 characters range to UTF-8.

**Parameters**
- **begin** Iterator pointing to the beginning of the input sequence
- **end** Iterator pointing to the end of the input sequence
- **output** Iterator pointing to the beginning of the output sequence
template<typename In, typename Out>

static Out sf::Utf<16>::toWide(In begin, In end, Out output, wchar_t replacement = 0)

Convert an UTF-16 characters range to wide characters.

Parameters

- **begin**: Iterator pointing to the beginning of the input sequence.
- **end**: Iterator pointing to the end of the input sequence.
- **output**: Iterator pointing to the beginning of the output sequence.
- **replacement**: Replacement for characters not convertible to wide (use 0 to skip them).

Returns

Iterator to the end of the output sequence which has been written.

The documentation for this class was generated from the following file:
- Utf.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doc
sf::Utf< 32 > Class Template Reference

Specialization of the Utf template for UTF-32. More...

#include <Utf.hpp>
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>template&lt;typename In&gt;</code></td>
<td><code>static In decode (In begin, In end, Uint32 &amp;output, Uint32 replacement=0)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Out&gt;</code></td>
<td><code>static Out encode (Uint32 input, Out output, Uint32 replacement=0)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In&gt;</code></td>
<td><code>static In next (In begin, In end)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In&gt;</code></td>
<td><code>static std::size_t count (In begin, In end)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out fromAnsi (In begin, In end, Out output, const std::locale &amp;locale=std::locale())</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out fromWide (In begin, In end, Out output)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out fromLatin1 (In begin, In end, Out output)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out toAnsi (In begin, In end, Out output, char replacement=0, const std::locale &amp;locale=std::locale())</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| `template<typename In , typename Out >
static Out toWide (In begin, In end, Out output, wchar_t replacement)` | Convert an UTF-32 characters range to wide characters.                       |
| `template<typename In , typename Out >
static Out toLatin1 (In begin, In end, Out output, char replacement)` | Convert an UTF-16 characters range to latin-1 (ISO-5589-1).                 |
| `template<typename In , typename Out >
static Out toUtf8 (In begin, In end, Out output)` | Convert a UTF-32 characters range to UTF-8.                                  |
| `template<typename In , typename Out >
static Out toUtf16 (In begin, In end, Out output)` | Convert a UTF-32 characters range to UTF-16.                                 |
| `template<typename In , typename Out >
static Out toUtf32 (In begin, In end, Out output)` | Convert a UTF-32 characters range to UTF-32.                                 |
| `template<typename In >
static Uint32 decodeAnsi (In input, const std::locale &locale=std::locale())` | Decode a single ANSI character to UTF-32.                                   |
| `template<typename In >
static Uint32 decodeWide (In input)` | Decode a single wide character to UTF-32.                                   |
| `template<typename Out >
static Out encodeAnsi (Uint32 codepoint, Out output, char replacement &locale=std::locale())` | Encode a single UTF-32 character to ANSI.                                   |
| `template<typename Out >
static Out encodeWide (Uint32 codepoint, Out output, wchar_t replacement)` | Encode a single UTF-32 character to wide.                                   |
Detailed Description

template<>  
class sf::Utf< 32 >

Specialization of the Utf template for UTF-32.

Definition at line 462 of file Utf.hpp.
Member Function Documentation

**template<typename In >**

```cpp
static std::size_t sf::Utf<32>::count ( In begin, In end )
```

Count the number of characters of a UTF-32 sequence.

This function is trivial for UTF-32, which can store every character in a si

**Parameters**

- **begin** Iterator pointing to the beginning of the input sequence
- **end** Iterator pointing to the end of the input sequence

**Returns**

Iterator pointing to one past the last read element of the input seque

**template<typename In >**

```cpp
static In sf::Utf<32>::decode ( In begin, In end, Uint32 & output, Uint32 replacement = 0 )
```

Decode a single UTF-32 character.

Decoding a character means finding its unique 32-bits code (called
standard. For UTF-32, the character value is the same as the codepoint.

**Parameters**

- **begin**  
  Iterator pointing to the beginning of the input sequence
- **end**  
  Iterator pointing to the end of the input sequence
- **output**  
  Codepoint of the decoded UTF-32 character
- **replacement**  
  Replacement character to use in case the UTF-8 sequence is invalid

**Returns**

Iterator pointing to one past the last read element of the input sequence

```cpp
template<typename In >
static Uint32 sf::Utf<32>::decodeAnsi ( In input, const std::locale & locale = )
```

Decode a single ANSI character to UTF-32.

This function does not exist in other specializations of sf::Utf<>, it is defined for convenience (it is used by several other conversion functions).

**Parameters**

- **input**  
  Input ANSI character
- **locale**  
  Locale to use for conversion

**Returns**

Converted character

```cpp
template<typename In >
static Uint32 sf::Utf<32>::decodeWide ( In input )
```
Decode a single wide character to UTF-32.

This function does not exist in other specializations of sf::Utf<>, it is defined for convenience (it is used by several other conversion functions).

**Parameters**

- **input** Input wide character

**Returns**

Converted character

```cpp
template<typename Out>
static Out sf::Utf<32>::encode(Uint32 input, Out output, Uint32 replacement = 0)
```

Encode a single UTF-32 character.

Encoding a character means converting a unique 32-bits code (called the codepoint) in the target encoding, UTF-32. For UTF-32, the codepoint is the same as the character value.

**Parameters**

- **input** Codepoint to encode as UTF-32
- **output** Iterator pointing to the beginning of the output sequence
- **replacement** Replacement for characters not convertible to UTF-32

**Returns**

Iterator to the end of the output sequence which has been written
template<typename Out >
static Out sf::Utf< 32 >::encodeAnsi ( Uint32 codepoint, Out output, char replacement, const std::locale & locale = std::locale() )

Encode a single UTF-32 character to ANSI.

This function does not exist in other specializations of sf::Utf<>; it is defined for convenience (it is used by several other conversion functions).

Parameters

codepoint
Iterator pointing to the beginning of the input sequence

type

output
Iterator pointing to the beginning of the output sequence

replacement
Replacement if the input character is not convertible to

locale
Locale to use for conversion

Returns

Iterator to the end of the output sequence which has been written

type

template<typename Out >
static Out sf::Utf< 32 >::encodeWide ( Uint32 codepoint, Out output, wchar_t replacement = \0 )

Encode a single UTF-32 character to wide.

This function does not exist in other specializations of sf::Utf<>; it is defined for convenience (it is used by several other conversion functions).
Parameters

- **codepoint**_iterator pointing to the beginning of the input sequence
- **output**_iterator pointing to the beginning of the output sequence
- **replacement**Replacement if the input character is not convertible to

Returns

Iterator to the end of the output sequence which has been written

---

### sf::Utf<32>::fromAnsi

```cpp
template<typename In , typename Out >
static Out sf::Utf< 32 >::fromAnsi ( In begin, In end, Out output, const std::locale & locale = std::locale() )
```

Convert an ANSI characters range to UTF-32.

The current global locale will be used by default, unless you pass a custom locale.

Parameters

- **begin**Iterator pointing to the beginning of the input sequence
- **end**Iterator pointing to the end of the input sequence
- **output**Iterator pointing to the beginning of the output sequence
- **locale**Locale to use for conversion

Returns

Iterator to the end of the output sequence which has been written

---

### sf::Utf<32>::fromLatin1

```cpp
template<typename In , typename Out >
static Out sf::Utf< 32 >::fromLatin1 ( In begin, In end, Out output, const std::locale & locale = std::locale() )
```
Convert a latin-1 (ISO-5589-1) characters range to UTF-32.

Parameters
- `begin` Iterator pointing to the beginning of the input sequence
- `end` Iterator pointing to the end of the input sequence
- `output` Iterator pointing to the beginning of the output sequence

Returns
- Iterator to the end of the output sequence which has been written

```cpp
template<typename In , typename Out >
static Out sf::Utf< 32 >::fromWide ( In begin, In end, Out output )
```

Convert a wide characters range to UTF-32.

Parameters
- `begin` Iterator pointing to the beginning of the input sequence
- `end` Iterator pointing to the end of the input sequence
- `output` Iterator pointing to the beginning of the output sequence

Returns
- Iterator to the end of the output sequence which has been written

```cpp
template<typename In >
static In sf::Utf< 32 >::next ( In begin,
```
Advance to the next UTF-32 character.

This function is trivial for UTF-32, which can store every character in a single storage element.

Parameters

- **begin**: Iterator pointing to the beginning of the input sequence
- **end**: Iterator pointing to the end of the input sequence

Returns

- Iterator pointing to one past the last read element of the input sequence

```cpp
template<typename In , typename Out >
static Out sf::Utf< 32 >::toAnsi ( In begin, In end, Out output, char replacement = 0 const std::locale & locale = std::locale() )
```

Convert an UTF-32 characters range to ANSI characters.

The current global locale will be used by default, unless you pass a custom one.

Parameters

- **begin**: Iterator pointing to the beginning of the input sequence
- **end**: Iterator pointing to the end of the input sequence
- **output**: Iterator pointing to the beginning of the output sequence
- **replacement**: Replacement for characters not convertible to ANSI (use 0 to skip them)
- **locale**: Locale to use for conversion
template<typename In , typename Out >

static Out sf::Utf< 32 >::toLatin1 ( In begin, In end, Out output, char replacement = 0 )

Convert an UTF-16 characters range to latin-1 (ISO-5589-1) characters.

Parameters
- **begin** Iterator pointing to the beginning of the input sequence
- **end** Iterator pointing to the end of the input sequence
- **output** Iterator pointing to the beginning of the output sequence
- **replacement** Replacement for characters not convertible to wide (use 0 to skip them)

Returns
Iterator to the end of the output sequence which has been written

template<typename In , typename Out >

static Out sf::Utf< 32 >::toUtf16 ( In begin, In end, Out output )

Convert a UTF-32 characters range to UTF-16.

Parameters
begin Iterator pointing to the beginning of the input sequence
end Iterator pointing to the end of the input sequence
output Iterator pointing to the beginning of the output sequence

Returns
Iterator to the end of the output sequence which has been written

template<typename In , typename Out >
static Out sf::Utf<32>::toUtf32 ( In begin, In end, Out output )

Convert a UTF-32 characters range to UTF-32.

This functions does nothing more than a direct copy; it is defined only to other specializations of the sf::Utf<> template, and allow generic code to

Parameters
begin Iterator pointing to the beginning of the input sequence
end Iterator pointing to the end of the input sequence
output Iterator pointing to the beginning of the output sequence

Returns
Iterator to the end of the output sequence which has been written

template<typename In , typename Out >
static Out sf::Utf<32>::toUtf8 ( In begin, In end, Out output )
Convert a UTF-32 characters range to UTF-8.

**Parameters**

- **begin** Iterator pointing to the beginning of the input sequence
- **end** Iterator pointing to the end of the input sequence
- **output** Iterator pointing to the beginning of the output sequence

**Returns**

Iterator to the end of the output sequence which has been written

```
template<typename In , typename Out >
static Out sf::Utf< 32 >::toWide ( In begin, In end,
Out output,
wchar_t replacement = 0 )
```

Convert an UTF-32 characters range to wide characters.

**Parameters**

- **begin** Iterator pointing to the beginning of the input sequence
- **end** Iterator pointing to the end of the input sequence
- **output** Iterator pointing to the beginning of the output sequence
- **replacement** Replacement for characters not convertible to wide (use 0 to skip them)

**Returns**

Iterator to the end of the output sequence which has been written

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

- Utf.hpp
sf::Utf< 8 > Class Template Reference

Specialization of the Utf template for UTF-8. More...

#include <Utf.hpp>
### Static Public Member Functions

**template<typename In >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static In decode (In begin, In end, Uint32 &amp;output, Uint32 replacement=0)</td>
<td>Decode a single UTF-8 character. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Out encode (Uint32 input, Out output, Uint8 replacement=0)</td>
<td>Encode a single UTF-8 character. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static In next (In begin, In end)</td>
<td>Advance to the next UTF-8 character. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static std::size_t count (In begin, In end)</td>
<td>Count the number of characters of a UTF-8 sequence.</td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Out fromAnsi (In begin, In end, Out output, const std::locale &amp;locale=std::locale())</td>
<td>Convert an ANSI characters range to UTF-8. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Out fromWide (In begin, In end, Out output)</td>
<td>Convert a wide characters range to UTF-8. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Out fromLatin1 (In begin, In end, Out output)</td>
<td>Convert a latin-1 (ISO-5589-1) characters range to UTF-8.</td>
</tr>
</tbody>
</table>

**template<typename In , typename Out >**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Out to Ansi (In begin, In end, Out output, char replacement=0, &amp;locale=std::locale())</td>
<td>Convert an UTF-8 characters range to ANSI characters.</td>
</tr>
<tr>
<td>Template</td>
<td>Function</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out toWide (In begin, In end, Out output, wchar_t replacement)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out toLatin1 (In begin, In end, Out output, char replacement)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out toUtf8 (In begin, In end, Out output)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out toUtf16 (In begin, In end, Out output)</code></td>
</tr>
<tr>
<td><code>template&lt;typename In, typename Out&gt;</code></td>
<td><code>static Out toUtf32 (In begin, In end, Out output)</code></td>
</tr>
</tbody>
</table>
Detailed Description

template<>  
class sf::Utf< 8 >

Specialization of the Utf template for UTF-8.

Definition at line 48 of file Utf.hpp.
Member Function Documentation

`template<typename In >
static std::size_t sf::Utf<8>::count ( In begin, In end )`

Count the number of characters of a UTF-8 sequence.

This function is necessary for multi-elements encodings, as a single character may use more than 1 storage element, thus the total size can be different from (begin - end).

**Parameters**
- `begin` Iterator pointing to the beginning of the input sequence
- `end` Iterator pointing to the end of the input sequence

**Returns**
- Iterator pointing to one past the last read element of the input sequence

`template<typename In >
static In sf::Utf<8>::decode ( In begin, In end, Uint32 & output, Uint32 replacement = 0 )`

Decode a single UTF-8 character.
Decoding a character means finding its unique 32-bits code (called the codepoint).

**Parameters**
- **begin**: Iterator pointing to the beginning of the input sequence.
- **end**: Iterator pointing to the end of the input sequence.
- **output**: Codepoint of the decoded UTF-8 character.
- **replacement**: Replacement character to use in case the UTF-8 sequence is invalid.

**Returns**
- Iterator pointing to one past the last read element of the input sequence.

```cpp
template<typename Out >
static Out sf::Utf< 8 >::encode ( Uint32 input, Out output, Uint8 replacement = 0 )
```

Encode a single UTF-8 character.

Encoding a character means converting a unique 32-bits code (called the codepoint) in the target encoding, UTF-8.

**Parameters**
- **input**: Codepoint to encode as UTF-8.
- **output**: Iterator pointing to the beginning of the output sequence.
- **replacement**: Replacement for characters not convertible to UTF-8 (0 to skip them).

**Returns**
- Iterator to the end of the output sequence which has been written.
template<typename In , typename Out >

static Out sf::Utf< 8 >::fromAnsi ( In begin, In end, Out output, 
const std::locale & locale = std::locale() )

Convert an ANSI characters range to UTF-8.

The current global locale will be used by default, unless you pass a custom one.

**Parameters**

- **begin**  
  Iterator pointing to the beginning of the input sequence

- **end**  
  Iterator pointing to the end of the input sequence

- **output**  
  Iterator pointing to the beginning of the output sequence

- **locale**  
  Locale to use for conversion

**Returns**

Iterator to the end of the output sequence which has been written

---

template<typename In , typename Out >

static Out sf::Utf< 8 >::fromLatin1 ( In begin, In end, Out output )

Convert a latin-1 (ISO-5589-1) characters range to UTF-8.

**Parameters**

- **begin**  
  Iterator pointing to the beginning of the input sequence

- **end**  
  Iterator pointing to the end of the input sequence

- **output**  
  Iterator pointing to the beginning of the output sequence
template<typename In , typename Out >

static Out sf::Utf< 8 >::fromWide ( In begin, In end, Out output )

Convert a wide characters range to UTF-8.

Parameters
  begin Iterator pointing to the beginning of the input sequence
  end Iterator pointing to the end of the input sequence
  output Iterator pointing to the beginning of the output sequence

Returns
  Iterator to the end of the output sequence which has been written

template<typename In >

static In sf::Utf< 8 >::next ( In begin, In end )

Advance to the next UTF-8 character.

This function is necessary for multi-elements encodings, as a single character may use more than 1 storage element.

Parameters
begin Iterator pointing to the beginning of the input sequence
end Iterator pointing to the end of the input sequence

Returns
Iterator pointing to one past the last read element of the input sequence

template<typename In , typename Out >
static Out sf::Utf< 8 >::toAnsi ( In begin, In end, Out output, char replacement = 0, const std::locale & locale = std::locale() )

Convert an UTF-8 characters range to ANSI characters.
The current global locale will be used by default, unless you pass a custom one.

Parameters
begin Iterator pointing to the beginning of the input sequence
end Iterator pointing to the end of the input sequence
output Iterator pointing to the beginning of the output sequence
replacement Replacement for characters not convertible to ANSI (use 0 to skip them)
locale Locale to use for conversion

Returns
Iterator to the end of the output sequence which has been written

template<typename In , typename Out >
static Out sf::Utf< 8 >::toLatin1 ( In begin, In end, Out output, char replacement = 0, const std::locale & locale = std::locale() )
template<typename In , typename Out >

static Out sf::Utf< 8 >::toUtf16 ( In begin, In end, Out output)

Convert a UTF-8 characters range to UTF-16.

Parameters

- **begin**: Iterator pointing to the beginning of the input sequence
- **end**: Iterator pointing to the end of the input sequence
- **output**: Iterator pointing to the beginning of the output sequence

Returns

- Iterator to the end of the output sequence which has been written
template<typename In , typename Out >

static Out sf::Utf<8>::toUtf32 ( In begin, In end, Out output )

Convert a UTF-8 characters range to UTF-32.

Parameters

**begin**  Iterator pointing to the beginning of the input sequence
**end**    Iterator pointing to the end of the input sequence
**output** Iterator pointing to the beginning of the output sequence

Returns

Iterator to the end of the output sequence which has been written

---

template<typename In , typename Out >

static Out sf::Utf<8>::toUtf8 ( In begin, In end, Out output )

Convert a UTF-8 characters range to UTF-8.

This functions does nothing more than a direct copy; it is defined only to allow other specializations of the sf::Utf<> template, and allow generic code to

Parameters

**begin**  Iterator pointing to the beginning of the input sequence
**end**    Iterator pointing to the end of the input sequence
**output** Iterator pointing to the beginning of the output sequence
Returns

Iterator to the end of the output sequence which has been written

```
template<typename In, typename Out>
static Out sf::Utf<8>::toWide ( In begin, In end, Out output, wchar_t replacement = 0 )
```

Convert an UTF-8 characters range to wide characters.

Parameters

- `begin` Iterator pointing to the beginning of the input sequence
- `end` Iterator pointing to the end of the input sequence
- `output` Iterator pointing to the beginning of the output sequence
- `replacement` Replacement for characters not convertible to wide (use 0 to skip them)

Returns

Iterator to the end of the output sequence which has been written

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

- `Utf.hpp`

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila :: Documentation generated by doxygen
Class Index

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R

A
- AlResource (sf)

B
- BlendMode (sf)

C
- SoundStream::Chunk (sf)
- CircleShape (sf)
- Clock (sf)
- Color (sf)
- Context (sf)
- ContextSettings (sf)
- ConvexShape (sf)
- Shader::CurrentTextureType (sf)

H
- Http (sf)

I
- Joystick::Identification (sf)
- Image (sf)
- SoundFileReader::Info (sf)
- Font::Info (sf)
- InputSoundFile (sf)
- InputStream (sf)
- IpAddress (sf)

J
- Joystick (sf)
- Event::JoystickButtonEvent (sf)
<table>
<thead>
<tr>
<th>D</th>
<th>Event::JoystickConnectEvent (sf)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Ftp::DirectoryResponse (sf)</td>
</tr>
<tr>
<td></td>
<td>Drawable (sf)</td>
</tr>
<tr>
<td>F</td>
<td>Event (sf)</td>
</tr>
<tr>
<td></td>
<td>FileInputStream (sf)</td>
</tr>
<tr>
<td></td>
<td>Font (sf)</td>
</tr>
<tr>
<td></td>
<td>Ftp (sf)</td>
</tr>
<tr>
<td>G</td>
<td>GlResource (sf)</td>
</tr>
<tr>
<td>M</td>
<td>MemoryInputStream (sf)</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
## Class Hierarchy

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

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf::AiResource</td>
<td>Base class for classes that require an OpenAL context</td>
</tr>
<tr>
<td>sf::SoundBuffer</td>
<td>Storage for audio samples defining a sound</td>
</tr>
<tr>
<td>sf::SoundRecorder</td>
<td>Abstract base class for capturing sound data</td>
</tr>
<tr>
<td>sf::SoundSource</td>
<td>Base class defining a sound</td>
</tr>
<tr>
<td>sf::BlendMode</td>
<td>Blending modes for drawing</td>
</tr>
<tr>
<td>sf::SoundStream::Chunk</td>
<td>Structure defining a chunk of audio data to stream</td>
</tr>
<tr>
<td>sf::Clock</td>
<td>Utility class that measures the elapsed time</td>
</tr>
<tr>
<td>sf::Color</td>
<td>Utility class for manipulating RGBA colors</td>
</tr>
<tr>
<td>sf::ContextSettings</td>
<td>Structure defining the settings of the OpenGL context attached to a window</td>
</tr>
<tr>
<td>sf::Shader::CurrentTextureType</td>
<td>Special type that can be passed to represents the texture of the object being drawn</td>
</tr>
<tr>
<td>sf::Drawable</td>
<td>Abstract base class for objects that can be drawn to a render target</td>
</tr>
<tr>
<td>sf::Shape</td>
<td>Base class for textured shapes</td>
</tr>
</tbody>
</table>
### sf::Sprite
Drawable representation of a texture, with its own transformations, color, etc.

### sf::Text
Graphical text that can be drawn to a render target.

### sf::VertexArray
Define a set of one or more 2D primitives.

### sf::Event
Defines a system event and its parameters.

### sf::Font
Class for loading and manipulating character fonts.

### sf::GlResource
Base class for classes that require an OpenGL context.

### sf::Context
Class holding a valid drawing context.

### sf::Shader
Shader class (vertex, geometry and fragment).

### sf::Texture
Image living on the graphics card that can be used for drawing.

### sf::Window
Window that serves as a target for OpenGL rendering.

### sf::Glyph
Structure describing a glyph.

### sf::Joystick::Identification
Structure holding a joystick's identification.

### sf::Image
Class for loading, manipulating and saving images.

### sf::SoundFileReader::Info
Structure holding the audio properties of a sound file.

### sf::Font::Info
Holds various information about a font.

### sf::InputStream
Abstract class for custom file input streams.

### sf::FileInputStream
Implementation of input stream based on a file.

### sf::MemoryInputStream
Implementation of input stream based on a memory chunk.

### sf::IpAddress
Encapsulate an IPv4 network address.

### sf::Joystick
Give access to the real-time state of the joysticks.

### sf::Event::JoystickButtonEvent
Joystick buttons events parameters (JoystickButtonDown, JoystickButtonReleased).

### sf::Event::JoystickConnectEvent
Joystick connection events parameters (JoystickConnected, JoystickDisconnected).

### sf::Event::JoystickMoveEvent
Joystick axis move event parameters.
- sf::Keyboard  
  Give access to the real-time state of the keyboard.

- sf::Event::KeyEvent  
  Keyboard event parameters (KeyPressed, KeyReleased)

- sf::Listener  
  The audio listener is the point in the scene from where the sounds are heard.

- sf::Mouse  
  Give access to the real-time state of the mouse.

- sf::Event::MouseButtonEvent  
  Mouse buttons events parameters (MouseButtonPressed, MouseButtonReleased)

- sf::Event::MouseMoveEvent  
  Mouse move event parameters

- sf::Event::MouseWheelEvent  
  Mouse wheel events parameters

- sf::Event::MouseWheelScrollEvent  
  Mouse wheel events parameters

- sf::NonCopyable  
  Utility class that makes any derived class non-copyable.

- sf::Context  
  Class holding a valid drawing context.

- sf::FileInputStream  
  Implementation of input stream based on a file.

- sf::Ftp  
  An FTP client.

- sf::GlResource::TransientContextLock  
  RAII helper class to temporarily lock an available context for use.

- sf::Http  
  A HTTP client.

- sf::InputSoundFile  
  Provide read access to sound files.

- sf::Lock  
  Automatic wrapper for locking and unlocking mutexes.

- sf::Mutex  
  Blocks concurrent access to shared resources from multiple threads.

- sf::OutputSoundFile  
  Provide write access to sound files.

- sf::RenderTarget  
  Base class for all render targets (window, texture, ...)

- sf::Shader  
  Shader class (vertex, geometry and fragment).

- sf::Socket  
  Base class for all the socket types.

- sf::Thread  
  Utility class to manipulate threads.

- sf::ThreadLocal  
  Defines variables with thread-local storage.
<table>
<thead>
<tr>
<th>sf::Window</th>
<th>Window that serves as a target for OpenGL rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf::Packet</td>
<td>Utility class to build blocks of data to transfer over the network</td>
</tr>
<tr>
<td>sf::Rect&lt; T &gt;</td>
<td>Utility class for manipulating 2D axis-aligned rectangles</td>
</tr>
<tr>
<td>sf::Rect&lt; float &gt;</td>
<td>Define the states used for drawing to a target</td>
</tr>
<tr>
<td>sf::Rect&lt; int &gt;</td>
<td>Define a HTTP response</td>
</tr>
<tr>
<td>sf::RenderStates</td>
<td>Define an HTTP request</td>
</tr>
<tr>
<td>sf::Http::Request</td>
<td>Define a FTP response</td>
</tr>
<tr>
<td>sf::Ftp::Response</td>
<td>Specialization of FTP response</td>
</tr>
<tr>
<td>sf::Ftp::DirectoryResponse</td>
<td>Specialization of FTP response</td>
</tr>
<tr>
<td>sf::Ftp::ListingResponse</td>
<td>Specialization of FTP response</td>
</tr>
<tr>
<td>sf::Http::Response</td>
<td>Define a HTTP response</td>
</tr>
<tr>
<td>sf::Sensor</td>
<td>Give access to the real-time state of the sensors</td>
</tr>
<tr>
<td>sf::Event::SensorEvent</td>
<td>Sensor event parameters (SensorChanged)</td>
</tr>
<tr>
<td>sf::Event::SizeEvent</td>
<td>Size events parameters (Resized)</td>
</tr>
<tr>
<td>sf::SocketSelector</td>
<td>Multiplexer that allows to read from multiple sockets</td>
</tr>
<tr>
<td>sf::SoundFileFactory</td>
<td>Manages and instantiates sound file readers and writers</td>
</tr>
<tr>
<td>sf::SoundFileReader</td>
<td>Abstract base class for sound file decoding</td>
</tr>
<tr>
<td>sf::SoundFileWriter</td>
<td>Abstract base class for sound file encoding</td>
</tr>
<tr>
<td>sf::String</td>
<td>Utility string class that automatically handles conversions between types and encodings</td>
</tr>
<tr>
<td>sf::Event::TextEvent</td>
<td>Text event parameters (TextEntered)</td>
</tr>
<tr>
<td>sf::Time</td>
<td>Represents a time value</td>
</tr>
<tr>
<td>sf::Touch</td>
<td>Give access to the real-time state of the touches</td>
</tr>
<tr>
<td>sf::Event::TouchEvent</td>
<td>Touch events parameters (Began, Moved, Ended)</td>
</tr>
<tr>
<td>sf::Transform</td>
<td>Define a 3x3 transform matrix</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sf::Transformable</td>
<td>Decomposed transform defined by a position, a rotation and a scale</td>
</tr>
<tr>
<td>sf::Shape</td>
<td>Base class for textured shapes</td>
</tr>
<tr>
<td>sf::Sprite</td>
<td>Drawable representation of a texture, with its own transformations, color, etc</td>
</tr>
<tr>
<td>sf::Text</td>
<td>Graphical text that can be drawn to a render target</td>
</tr>
<tr>
<td>sf::Utf&lt; N &gt;</td>
<td>Utility class providing generic functions for UTF conversions</td>
</tr>
<tr>
<td>sf::Utf&lt; 16 &gt;</td>
<td>Specialization of the Utf template for UTF-16</td>
</tr>
<tr>
<td>sf::Utf&lt; 32 &gt;</td>
<td>Specialization of the Utf template for UTF-32</td>
</tr>
<tr>
<td>sf::Utf&lt; 8 &gt;</td>
<td>Specialization of the Utf template for UTF-8</td>
</tr>
<tr>
<td>sf::Vector2&lt; T &gt;</td>
<td>Utility template class for manipulating 2-dimensional vectors</td>
</tr>
<tr>
<td>sf::Vector2&lt; float &gt;</td>
<td></td>
</tr>
<tr>
<td>sf::Vector2&lt; unsigned int &gt;</td>
<td></td>
</tr>
<tr>
<td>sf::Vector3&lt; T &gt;</td>
<td>Utility template class for manipulating 3-dimensional vectors</td>
</tr>
<tr>
<td>sf::Vertex</td>
<td>Define a point with color and texture coordinates</td>
</tr>
<tr>
<td>sf::VideoMode</td>
<td>VideoMode defines a video mode</td>
</tr>
<tr>
<td>sf::View</td>
<td>2D camera that defines what region is shown on screen</td>
</tr>
</tbody>
</table>
Here is a list of all documented class members with links to the class documentation:

- a -

- a : sf::Color
- A : sf::Keyboard
- Accelerometer : sf::Sensor
- accept() : sf::TcpListener
- Accepted : sf::Http::Response
- Add : sf::BlendMode, sf::Keyboard
- add() : sf::SocketSelector
- advance : sf::Glyph
- alphaDstFactor : sf::BlendMode
- alphaEquation : sf::BlendMode
- alphaSrcFactor : sf::BlendMode
- AlResource() : sf::AlResource
- alt : sf::Event::KeyEvent
- antialiasingLevel : sf::ContextSettings
- Any : sf::IpAddress
- AnyPort : sf::Socket
- append() : sf::Packet , sf::VertexArray
- Ascii : sf::Ftp
- asMicroseconds() : sf::Time
- asMilliseconds() : sf::Time
- asSeconds() : sf::Time
- Attribute : sf::ContextSettings
- attributeFlags : sf::ContextSettings
- axis : sf::Event::JoystickMoveEvent
- Axis : sf::Joystick
- AxisCount : sf::Joystick
Here is a list of all documented class members with links to the class documentation:

- **b** -

- b : sf::Color
- B : sf::Keyboard
- BackSlash : sf::Keyboard
- BackSpace : sf::Keyboard
- BadCommandSequence : sf::Ftp::Response
- BadGateway : sf::Http::Response
- BadRequest : sf::Http::Response
- begin() : sf::String
- Binary : sf::Ftp
- bind() : sf::Shader, sf::Texture, sf::UdpSocket
- bitsPerPixel : sf::VideoMode
- Black : sf::Color
- BlendMode() : sf::BlendMode
- blendMode : sf::RenderStates
- Blue : sf::Color
- Bold : sf::Text
- bounds : sf::Glyph
- Broadcast : sf::IpAddress
- button : sf::Event::JoystickButtonEvent, sf::Event::MouseButtonEvent
- Button : sf::Mouse
- ButtonCount : sf::Joystick, sf::Mouse
Here is a list of all documented class members with links to the class documentation:

- C -

- C : sf::Keyboard
- capture() : sf::RenderWindow
- changeDirectory() : sf::Ftp
- channelCount : sf::SoundFileReader::Info
- CircleShape() : sf::CircleShape
- clear() : sf::Packet, sf::RenderTarget, sf::SocketSelector, sf::String,
- Clock() : sf::Clock
- close() : sf::Socket, sf::TcpListener, sf::Window
- Closed : sf::Event
- ClosingConnection : sf::Ftp::Response
- ClosingDataConnection : sf::Ftp::Response
- code : sf::Event::KeyEvent
- Color() : sf::Color
- color : sf::Vertex
- `colorDstFactor : sf::BlendMode`
- `colorEquation : sf::BlendMode`
- `colorSrcFactor : sf::BlendMode`
- `combine() : sf::Transform`
- `Comma : sf::Keyboard`
- `CommandNotImplemented : sf::Ftp::Response`
- `CommandUnknown : sf::Ftp::Response`
- `connect() : sf::Ftp, sf::TcpSocket`
- `ConnectionClosed : sf::Ftp::Response`
- `ConnectionFailed : sf::Ftp::Response, sf::Http::Response`
- `ConstIterator : sf::String`
- `contains() : sf::Rect< T >`
- `Context() : sf::Context`
- `ContextSettings() : sf::ContextSettings`
- `control : sf::Event::KeyEvent`
- `ConvexShape() : sf::ConvexShape`
- `CoordinateType : sf::Texture`
- `copy() : sf::Image`
- `copyToImage() : sf::Texture`
- `Core : sf::ContextSettings`
- `Count : sf::Event, sf::Joystick, sf::Sensor`
- `count() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >`
- `create() : sf::Image, sf::RenderTexture, sf::Socket, sf::Texture, sf::Window`
- `Created : sf::Http::Response`
- `createDirectory() : sf::Ftp`
- `createMaskFromColor() : sf::Image`
- createReaderFromFilename() : sf::SoundFileFactory
- createReaderFromMemory() : sf::SoundFileFactory
- createReaderFromStream() : sf::SoundFileFactory
- createWriterFromFilename() : sf::SoundFileFactory
- CurrentTexture : sf::Shader
- Cyan : sf::Color
Here is a list of all documented class members with links to the class documentation:

- **D** : sf::Keyboard
  - Dash : sf::Keyboard
  - DataConnectionAlreadyOpened : sf::Ftp::Response
  - DataConnectionOpened : sf::Ftp::Response
  - DataConnectionUnavailable : sf::Ftp::Response
  - Debug : sf::ContextSettings
  - decode() : sf::Utf< 16 > , sf::Utf< 32 > , sf::Utf< 8 >
  - decodeAnsi() : sf::Utf< 32 >
  - decodeWide() : sf::Utf< 32 >
  - Default : sf::ContextSettings , sf::RenderStates
  - Delete : sf::Http::Request , sf::Keyboard
  - deleteDirectory() : sf::Ftp
  - deleteFile() : sf::Ftp
  - delta : sf::Event::MouseWheelEvent , sf::Event::MouseWheelScrollEvent
- depthBits : sf::ContextSettings
- DirectoryOk : sf::Ftp::Response
- DirectoryResponse() : sf::Ftp::DirectoryResponse
- DirectoryStatus : sf::Ftp::Response
- disconnect() : sf::Ftp, sf::TcpSocket
- Disconnected : sf::Socket
- display() : sf::RenderTexture, sf::Window
- Divide : sf::Keyboard
- Done : sf::Socket
- Down : sf::Keyboard
- download() : sf::Ftp
- draw() : sf::Drawable, sf::RenderTarget
- DstAlpha : sf::BlendMode
- DstColor : sf::BlendMode

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
Here is a list of all documented class members with links to the class documentation:

- e -

- E : sf::Keyboard
- Ebcdic : sf::Ftp
- encode() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >
- encodeAnsi() : sf::Utf< 32 >
- encodeWide() : sf::Utf< 32 >
- End : sf::Keyboard
- end() : sf::String
- endOfPacket() : sf::Packet
- ensureGlContext() : sf::GlResource
- EnteringPassiveMode : sf::Ftp::Response
- Equal : sf::Keyboard
- Equation : sf::BlendMode
- erase() : sf::String
- Error : sf::Socket
- Escape: sf::Keyboard
- EventType: sf::Event
Here is a list of all documented class members with links to the class documentation:

- f -

- F : sf::Keyboard
- F1 : sf::Keyboard
- F10 : sf::Keyboard
- F11 : sf::Keyboard
- F12 : sf::Keyboard
- F13 : sf::Keyboard
- F14 : sf::Keyboard
- F15 : sf::Keyboard
- F2 : sf::Keyboard
- F3 : sf::Keyboard
- F4 : sf::Keyboard
- F5 : sf::Keyboard
- F6 : sf::Keyboard
- F7 : sf::Keyboard
- F8 : sf::Keyboard
- F9 : sf::Keyboard
- Factor : sf::BlendMode
- family : sf::Font::Info
- FileActionAborted : sf::Ftp::Response
- FileActionOk : sf::Ftp::Response
- FileInputStream() : sf::FileInputStream
- FilenameNotAllowed : sf::Ftp::Response
- FileStatus : sf::Ftp::Response
- FileUnavailable : sf::Ftp::Response
- find() : sf::String
- findCharacterPos() : sf::Text
- finger : sf::Event::TouchEvent
- flipHorizontally() : sf::Image
- flipVertically() : sf::Image
- Font() : sf::Font
- Forbidden : sf::Http::Response
- Fragment : sf::Shader
- fromAnsi() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >
- fromLatin1() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >
- fromUtf16() : sf::String
- fromUtf32() : sf::String
- fromUtf8() : sf::String
- fromWide() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >

SFML is licensed under the terms and conditions of the zlib/png license.
Here is a list of all documented class members with links to the class documentation:

- **g** -

- g : sf::Color
- G : sf::Keyboard
- GainedFocus : sf::Event
- GatewayTimeout : sf::Http::Response
- generateMipmap() : sf::RenderTexture, sf::Texture
- Geometry : sf::Shader
- Get : sf::Http::Request
- getActiveContext() : sf::Context
- getAttenuation() : sf::SoundSource
- getAvailableDevices() : sf::SoundRecorder
- getAxisPosition() : sf::Joystick
- getBody() : sf::Http::Response
- getBounds() : sf::VertexArray
- getBuffer() : sf::Sound, sf::SoundBufferRecorder
- getButtonCount() : sf::Joystick
- getCenter() : sf::View
- getChannelCount() : sf::InputSoundFile , sf::SoundBuffer , sf::SoundRecorder
- getCharacterSize() : sf::Text
- getColor() : sf::Sprite , sf::Text
- getData() : sf::Packet , sf::String
- getDataSize() : sf::Packet
- getDefaultDevice() : sf::SoundRecorder
- getDefaultView() : sf::RenderTarget
- getDesktopMode() : sf::VideoMode
- getDevice() : sf::SoundRecorder
- getDirection() : sf::Listener
- getDirectory() : sf::Ftp::DirectoryResponse
- getDirectoryListing() : sf::Ftp
- getDuration() : sf::InputSoundFile , sf::Music , sf::SoundBuffer
- getElapsedTime() : sf::Clock
- getField() : sf::Http::Response
- getFillColor() : sf::Shape , sf::Text
- getFont() : sf::Text
- getFullscreenModes() : sf::VideoMode
- getFunction() : sf::Context
- getGlobalBounds() : sf::Shape , sf::Sprite , sf::Text
- getGlobalVolume() : sf::Listener
- getGlyph() : sf::Font
- getHandle() : sf::Socket
- getIdentification() : sf::Joystick
- `getInfo()` : `sf::Font`
- `getInverse()` : `sf::Transform`
- `getInverseTransform()` : `sf::Transformable`, `sf::View`
- `getKerning()` : `sf::Font`
- `getLineSpacing()` : `sf::Font`
- `getListing()` : `sf::Ftp::ListingResponse`
- `getLocalAddress()` : `sf::IpAddress`
- `getLocalBounds()` : `sf::Shape`, `sf::Sprite`, `sf::Text`
- `getLocalPort()` : `sf::TcpListener`, `sf::TcpSocket`, `sf::UdpSocket`
- `getLoop()` : `sf::Sound`, `sf::SoundStream`
- `getMajorHttpVersion()` : `sf::Http::Response`
- `getMatrix()` : `sf::Transform`
- `getMaximumSize()` : `sf::Texture`
- `getMessage()` : `sf::Ftp::Response`
- `getMinDistance()` : `sf::SoundSource`
- `getMinorHttpVersion()` : `sf::Http::Response`
- `getNativeHandle()` : `sf::Shader`, `sf::Texture`
- `getOrigin()` : `sf::Transformable`
- `getOutlineColor()` : `sf::Shape`, `sf::Text`
- `getOutlineThickness()` : `sf::Shape`, `sf::Text`
- `getPitch()` : `sf::SoundSource`
- `getPixel()` : `sf::Image`
- `getPixelsPtr()` : `sf::Image`
- `getPlayingOffset()` : `sf::Sound`, `sf::SoundStream`
- `getPoint()` : `sf::CircleShape`, `sf::ConvexShape`, `sf::RectangleShape`
- `getPointCount()` : `sf::CircleShape`, `sf::ConvexShape`, `sf::RectangleShape`
- `getPosition()` : `sf::Listener`, `sf::Mouse`, `sf::SoundSource`, `sf::Touch`, ...
- `getPrimitiveType()` : `sf::VertexArray`
- `getPublicAddress()` : `sf::IpAddress`
- `getRadius()` : `sf::CircleShape`
- `getRemoteAddress()` : `sf::TcpSocket`
- `getRemotePort()` : `sf::TcpSocket`
- `getRotation()` : `sf::Transformable`, `sf::View`
- `getSampleCount()` : `sf::InputSoundFile`, `sf::SoundBuffer`
- `getSampleRate()` : `sf::InputSoundFile`, `sf::SoundBuffer`, `sf::SoundRecorder`
- `getSamples()` : `sf::SoundBuffer`
- `getScale()` : `sf::Transformable`
- `getSettings()` : `sf::Context`, `sf::Window`
- `getSize()` : `sf::FileInputStream`, `sf::Image`, `sf::InputStream`, `sf::Memo`, `sf::RectangleShape`, `sf::RenderTarget`, `sf::RenderTexture`, `sf::RenderTexture`, `sf::View`, `sf::Window`
- `getStatus()` : `sf::Ftp::Response`, `sf::Http::Response`, `sf::Sound`, `sf::SoundSource`
- `getString()` : `sf::Text`
- `getStyle()` : `sf::Text`
- `getSystemHandle()` : `sf::Window`
- `getText()` : `sf::Font`, `sf::RenderTexture`, `sf::Shape`, `sf::Sprite`
- `getTexturedRect()` : `sf::Shape`, `sf::Sprite`
- `getTransform()` : `sf::Transformable`, `sf::View`
- `getUnderlinePosition()` : `sf::Font`
- `getUnderlineThickness()` : `sf::Font`
- `getUpVector()` : `sf::Listener`
- `getValue()` : `sf::Sensor`, `sf::ThreadLocal`
- `getVertexCount()` : `sf::VertexArray`
- getView() : sf::RenderTarget
- getViewport() : sf::RenderTarget, sf::View
- getVolume() : sf::SoundSource
- getWorkingDirectory() : sf::Ftp
- GlResource() : sf::GlResource
- Glyph() : sf::Glyph
- Gravity : sf::Sensor
- Green : sf::Color
- Gyroscope : sf::Sensor
Here is a list of all documented class members with links to the class documentation:

- **h** -

- **H** : sf::Keyboard
- **hasAxis()** : sf::Joystick
- **hasFocus()** : sf::Window
- **Head** : sf::Http::Request
- **height** : sf::Event::SizeEvent, sf::Rect< T >, sf::VideoMode
- **HelpMessage** : sf::Ftp::Response
- **Home** : sf::Keyboard
- **HorizontalWheel** : sf::Mouse
- **Http()** : sf::Http
Here is a list of all documented class members with links to the class documentation:

- i -

- I : sf::Keyboard
- Identity : sf::Transform
- Image() : sf::Image
- initialize() : sf::RenderTarget, sf::SoundStream
- InputSoundFile() : sf::InputSoundFile
- Insert : sf::Keyboard
- insert() : sf::String
- InsufficientStorageSpace : sf::Ftp::Response
- InternalServerError : sf::Http::Response
- intersects() : sf::Rect<T>
- InvalidFile : sf::Ftp::Response
- InvalidPos : sf::String
- InvalidResponse : sf::Ftp::Response, sf::Http::Response
- IpAddress() : sf::IpAddress
- isAvailable() : sf::Sensor, sf::Shader, sf::SoundRecorder
- isBlocking() : sf::Socket
- isButtonPressed() : sf::Joystick, sf::Mouse
- isConnected() : sf::Joystick
- isDown() : sf::Touch
- isEmpty() : sf::String
- isExtensionAvailable() : sf::Context
- isGeometryAvailable() : sf::Shader
- isKeyPressed() : sf::Keyboard
-isOk() : sf::Ftp::Response
- isOpen() : sf::Window
- isReady() : sf::SocketSelector
- isRelativeToListener() : sf::SoundSource
- isRepeated() : sf::RenderTexture, sf::Texture
- isSmooth() : sf::RenderTexture, sf::Texture
- isSrgb() : sf::Texture
- isValid() : sf::VideoMode
- Italic : sf::Text
- Iterator : sf::String
Here is a list of all documented class members with links to the class documentation:

- **j** -

- J : sf::Keyboard
- joystickButton : sf::Event
- JoystickButtonPressed : sf::Event
- JoystickButtonReleased : sf::Event
- joystickConnect : sf::Event
- JoystickConnected : sf::Event
- JoystickDisconnected : sf::Event
- joystickId : sf::Event::JoystickButtonEvent, sf::Event::JoystickConnectEvent, sf::Event::JoystickMoveEvent
- joystickMove : sf::Event
- JoystickMoved : sf::Event
Here is a list of all documented class members with links to the class documentation:

- k -

- K : sf::Keyboard
- keepAlive() : sf::Ftp
- key : sf::Event
- Key : sf::Keyboard
- KeyCount : sf::Keyboard
- KeyPressed : sf::Event
- KeyReleased : sf::Event
Here is a list of all documented class members with links to the class documentation:

- L : sf::Keyboard
- LAlt : sf::Keyboard
- launch() : sf::Thread
- LBracket : sf::Keyboard
- LControl : sf::Keyboard
- Left : sf::Keyboard, sf::Mouse
- left : sf::Rect< T >
- listen() : sf::TcpListener
- ListingResponse() : sf::Ftp::ListingResponse
- loadFromFile() : sf::Font, sf::Image, sf::Shader, sf::SoundBuffer, sf:
- loadFromImage() : sf::Texture
- loadFromMemory() : sf::Font, sf::Image, sf::Shader, sf::SoundBuffer
- loadFromSamples() : sf::SoundBuffer
- loadFromStream() : sf::Font, sf::Image, sf::Shader, sf::SoundBuffer
- LocalError : sf::Ftp::Response
- LocalHost : sf::IpAddress
- Lock() : sf::Lock
- lock() : sf::Mutex
- LoggedIn : sf::Ftp::Response
- login() : sf::Ftp
- LostFocus : sf::Event
- LShift : sf::Keyboard
- LSystem : sf::Keyboard

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
Here is a list of all documented class members with links to the class documentation:

- **m** -

  - M : sf::Keyboard
  - m_source : sf::SoundSource
  - Magenta : sf::Color
  - Magnetometer : sf::Sensor
  - majorVersion : sf::ContextSettings
  - mapCoordsToPixel() : sf::RenderTarget
  - mapPixelToCoords() : sf::RenderTarget
  - MaxDatagramSize : sf::UdpSocket
  - MemoryInputStream() : sf::MemoryInputStream
  - Menu : sf::Keyboard
  - Method : sf::Http::Request
  - microseconds() : sf::Time
  - Middle : sf::Mouse
  - milliseconds() : sf::Time
- minorVersion : sf::ContextSettings
- mouseButton : sf::Event
- MouseButtonPressed : sf::Event
- MouseButtonReleased : sf::Event
- MouseEntered : sf::Event
- MouseLeft : sf::Event
- mouseMove : sf::Event
- MouseMoved : sf::Event
- mouseWheel : sf::Event
- MouseWheelMoved : sf::Event
- mouseWheelScroll : sf::Event
- MouseWheelScrolled : sf::Event
- move() : sf::Transformable, sf::View
- MovedPermanently : sf::Http::Response
- MovedTemporarily : sf::Http::Response
- MultipleChoices : sf::Http::Response
- Multiply : sf::Keyboard
- Music() : sf::Music
- Mutex() : sf::Mutex
Here is a list of all documented class members with links to the class documentation:

- n -

- N : sf::Keyboard
- name : sf::Joystick::Identification
- NeedAccountToLogIn : sf::Ftp::Response
- NeedAccountToStore : sf::Ftp::Response
- NeedInformation : sf::Ftp::Response
- NeedPassword : sf::Ftp::Response
- next() : sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- NoContent : sf::Http::Response
- NonCopyable() : sf::NonCopyable
- None : sf::IpAddress
-Normalized : sf::Texture
- NotEnoughMemory : sf::Ftp::Response
- NotFound : sf::Http::Response
- NotImplemented : sf::Http::Response
- NotLoggedIn : sf::Ftp::Response
- NotModified : sf::Http::Response
- NotReady : sf::Socket
- Num0 : sf::Keyboard
- Num1 : sf::Keyboard
- Num2 : sf::Keyboard
- Num3 : sf::Keyboard
- Num4 : sf::Keyboard
- Num5 : sf::Keyboard
- Num6 : sf::Keyboard
- Num7 : sf::Keyboard
- Num8 : sf::Keyboard
- Num9 : sf::Keyboard
- Numpad0 : sf::Keyboard
- Numpad1 : sf::Keyboard
- Numpad2 : sf::Keyboard
- Numpad3 : sf::Keyboard
- Numpad4 : sf::Keyboard
- Numpad5 : sf::Keyboard
- Numpad6 : sf::Keyboard
- Numpad7 : sf::Keyboard
- Numpad8 : sf::Keyboard
- Numpad9 : sf::Keyboard

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by


Here is a list of all documented class members with links to the class documentation:

- **O** -

- **O**: sf::Keyboard
- **Ok**: sf::Ftp::Response, sf::Http::Response
- **onCreate()**: sf::RenderWindow, sf::Window
- **One**: sf::BlendMode
- **OneMinusDstAlpha**: sf::BlendMode
- **OneMinusDstColor**: sf::BlendMode
- **OneMinusSrcAlpha**: sf::BlendMode
- **OneMinusSrcColor**: sf::BlendMode
- **onGetData()**: sf::Music, sf::SoundStream
- **onProcessSamples()**: sf::SoundBufferRecorder, sf::SoundRecorder
- **onReceive()**: sf::Packet
- **onResize()**: sf::RenderWindow, sf::Window
- **onSeek()**: sf::Music, sf::SoundStream
- **onSend()**: sf::Packet
- `onStart()`: `sf::SoundBufferRecorder`, `sf::SoundRecorder`
- `onStop()`: `sf::SoundBufferRecorder`, `sf::SoundRecorder`
- `open()`: `sf::FileInputStream`, `sf::MemoryInputStream`, `sf::SoundFileReader`
- `openForWriting()`: `sf::InputSoundFile`
- `openFromFile()`: `sf::InputSoundFile`, `sf::Music`, `sf::OutputSoundFile`
- `openFromMemory()`: `sf::InputSoundFile`, `sf::Music`
- `openFromStream()`: `sf::InputSoundFile`, `sf::Music`
- `OpeningDataConnection`: `sf::Ftp::Response`
- `operator BoolType()`: `sf::Packet`
- `operator std::string()`: `sf::String`
- `operator std::wstring()`: `sf::String`
- `operator T *()`: `sf::ThreadLocalPtr<T>`
- `operator!=( )`: `sf::BlendMode`, `sf::Color`, `sf::Rect<T>`, `sf::String`, `sf::Vector3<T>`, `sf::VideoMode`
- `operator%()`: `sf::Time`
- `operator %=()`: `sf::Time`
- `operator*()`: `sf::Color`, `sf::ThreadLocalPtr<T>`, `sf::Time`, `sf::Transform`, `sf::Vector3<T>`
- `operator*=()`: `sf::Color`, `sf::Time`, `sf::Transform`, `sf::Vector2<T>`, `sf::Vector3<T>`
- `operator+=()`: `sf::Color`, `sf::String`, `sf::Time`, `sf::Vector2<T>`, `sf::Vector3<T>`
- `operator-=()`: `sf::Color`, `sf::Time`, `sf::Vector2<T>`, `sf::Vector3<T>`
- `operator->()`: `sf::ThreadLocalPtr<T>`
- `operator/()`: `sf::Time`, `sf::Vector2<T>`, `sf::Vector3<T>`
- `operator/()`: `sf::Time`, `sf::Vector2<T>`, `sf::Vector3<T>`
- `operator<`: `sf::IpAddress`, `sf::String`, `sf::Time`, `sf::VideoMode`
- `operator<<() : sf::Packet`
- `operator<=() : sf::String, sf::Time, sf::VideoMode`
- `operator=() : sf::Font, sf::SocketSelector, sf::Sound, sf::SoundBuffer, sf::Texture, sf::ThreadLocalPtr<T>`
- `operator==() : sf::BlendMode, sf::Color, sf::Rect<T>, sf::String, sf::Vector3<T>, sf::VideoMode`
- `operator>() : sf::String, sf::Time, sf::VideoMode`
- `operator>=() : sf::String, sf::Time, sf::VideoMode`
- `operator>>() : sf::Packet`
- `operator[]() : sf::String, sf::VertexArray`
- `Orientation : sf::Sensor`
- `OutputSoundFile() : sf::OutputSoundFile`

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
**Class Members**

Here is a list of all documented class members with links to the class documentation:

- **p** -

- P : sf::Keyboard
- Packet() : sf::Packet
- PageDown : sf::Keyboard
- PageTypeUnknown : sf::Ftp::Response
- PageUp : sf::Keyboard
- ParameterNotImplemented : sf::Ftp::Response
- ParametersUnknown : sf::Ftp::Response
- parentDirectory() : sf::Ftp
- Partial : sf::Socket
- PartialContent : sf::Http::Response
- Pause : sf::Keyboard
- pause() : sf::Sound, sf::SoundStream
- Paused : sf::SoundSource
- Period : sf::Keyboard
- Pixels : sf::Texture
- play() : sf::Sound, sf::SoundStream
- Playing : sf::SoundSource
- PointlessCommand : sf::Ftp::Response
- pollEvent() : sf::Window
- popGLStates() : sf::RenderTarget
- position : sf::Event::JoystickMoveEvent, sf::Vertex
- Post : sf::Http::Request
- PovX : sf::Joystick
- PovY : sf::Joystick
- productId : sf::Joystick::Identification
- pushGLStates() : sf::RenderTarget
- Put : sf::Http::Request
Here is a list of all documented class members with links to the class documentation:

- **Q**: sf::Keyboard
- **Quote**: sf::Keyboard
Here is a list of all documented class members with links to the class documentation:

- r -

- r : sf::Color
- R : sf::Joystick, sf::Keyboard
- RAlt : sf::Keyboard
- RangeNotSatisfiable : sf::Http::Response
- RBracket : sf::Keyboard
- RControl : sf::Keyboard
- read() : sf::FileInputStream, sf::InputSoundFile, sf::InputStream, sf::InputStream
  sf::SoundFileReader
- receive() : sf::TcpSocket, sf::UdpSocket
- Rect() : sf::Rect<T>
- RectangleShape() : sf::RectangleShape
- Red : sf::Color
- registerReader() : sf::SoundFileFactory
- registerWriter() : sf::SoundFileFactory
- Regular : sf::Text
- remove() : sf::SocketSelector
- renameFile() : sf::Ftp
- RenderStates() : sf::RenderStates
- RenderTarget() : sf::RenderTarget
- RenderTexture() : sf::RenderTexture
- RenderWindow() : sf::RenderWindow
- replace() : sf::String
- Request() : sf::Http::Request
- requestFocus() : sf::Window
- reset() : sf::View
- resetBuffer() : sf::Sound
- ResetContent : sf::Http::Response
- resetGLStates() : sf::RenderTarget
- resize() : sf::VertexArray
- Resized : sf::Event
- Response() : sf::Ftp::Response, sf::Http::Response
- restart() : sf::Clock
- RestartMarkerReply : sf::Ftp::Response
- Return : sf::Keyboard
- ReverseSubtract : sf::BlendMode
- Right : sf::Keyboard, sf::Mouse
- rotate() : sf::Transform, sf::Transformable, sf::View
- RShift : sf::Keyboard
- RSysstem : sf::Keyboard
SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
Here is a list of all documented class members with links to the class documentation:

- **S** -
  - S : sf::Keyboard
  - sampleCount : sf::SoundFileReader::Info, sf::SoundStream::Chunk
  - sampleRate : sf::SoundFileReader::Info
  - samples : sf::SoundStream::Chunk
  - saveToFile() : sf::Image, sf::SoundBuffer
  - scale() : sf::Transform, sf::Transformable
  - seconds() : sf::Time
  - seek() : sf::FileInputStream, sf::InputSoundFile, sf::InputStream, sf::SoundFileReader
  - SemiColon : sf::Keyboard
  - send() : sf::TcpSocket, sf::UdpSocket
  - sendCommand() : sf::Ftp
  - sendRequest() : sf::Http
  - sensor : sf::Event
- SensorChanged : sf::Event
- ServiceNotAvailable : sf::Http::Response
- ServiceReady : sf::Ftp::Response
- ServiceReadySoon : sf::Ftp::Response
- ServiceUnavailable : sf::Ftp::Response
- setActive() : sf::Context, sf::RenderTexture, sf::Window
- setAttenuation() : sf::SoundSource
- setBlocking() : sf::Socket
- setBody() : sf::Http::Request
- setBuffer() : sf::Sound
- setCenter() : sf::View
- setChannelCount() : sf::SoundRecorder
- setCharacterSize() : sf::Text
- setColor() : sf::Sprite, sf::Text
- setDevice() : sf::Sprite, sf::Text
- setDirection() : sf::Listener
- setEnabled() : sf::Sensor
- setField() : sf::Http::Request
- setFillColor() : sf::Shape, sf::Text
- setFont() : sf::Text
- setFramerateLimit() : sf::Window
- setGlobalVolume() : sf::Listener
- setHost() : sf::Http
- setHttpVersion() : sf::Http::Request
- setIcon() : sf::Window
- setJoystickThreshold() : sf::Window
- `setKeyRepeatEnabled()` : sf::Window
- `setLoop()` : sf::Sound, sf::SoundStream
- `setMethod()` : sf::Http::Request
- `setMinDistance()` : sf::SoundSource
- `setMouseCursorGrabbed()` : sf::Window
- `setMouseCursorVisible()` : sf::Window
- `setOrigin()` : sf::Transformable
- `setOutlineColor()` : sf::Shape, sf::Text
- `setOutlineThickness()` : sf::Shape, sf::Text
- `setParameter()` : sf::Shader
- `setPitch()` : sf::SoundSource
- `setPixel()` : sf::Image
- `setPlayingOffset()` : sf::Sound, sf::SoundStream
- `setPoint()` : sf::ConvexShape
- `setPointCount()` : sf::CircleShape, sf::ConvexShape
- `setPosition()` : sf::Listener, sf::Mouse, sf::SoundSource, sf::Transformable
- `setPrimitiveType()` : sf::VertexArray
- `setProcessingInterval()` : sf::SoundRecorder
- `setRadius()` : sf::CircleShape
- `setRelativeToListener()` : sf::SoundSource
- `setRepeated()` : sf::RenderTexture, sf::Texture
- `setRotation()` : sf::RenderTexture, sf::Texture
- `setScale()` : sf::Transformable
- `setSize()` : sf::RectangleShape, sf::View, sf::Window
- `setSmooth()` : sf::RenderTexture, sf::Texture
- `setSrgb()` : sf::Texture
- `string()` : `sf::Text`
- `style()` : `sf::Text`
- `texture()` : `sf::Shape`, `sf::Sprite`
- `textureRect()` : `sf::Shape`, `sf::Sprite`
- `title()` : `sf::Window`
- `uniform()` : `sf::Shader`
- `uniformArray()` : `sf::Shader`
- `upVector()` : `sf::Listener`
- `uri()` : `sf::Http::Request`
- `value()` : `sf::ThreadLocal`
- `verticalSyncEnabled()` : `sf::Window`
- `view()` : `sf::RenderTarget`
- `viewport()` : `sf::View`
- `virtualKeyboardVisible()` : `sf::Keyboard`
- `visible()` : `sf::Window`
- `volume()` : `sf::SoundSource`
- `shader` : `sf::RenderStates`
- `shader()` : `sf::Shader`
- `shape()` : `sf::Shape`
- `shift` : `sf::Event::KeyEvent`
- `size` : `sf::Event`
- `slash` : `sf::Keyboard`
- `socket()` : `sf::Socket`
- `socketSelector()` : `sf::SocketSelector`
- `sound()` : `sf::Sound`
- `soundBuffer()` : `sf::SoundBuffer`
- SoundRecorder() : sf::SoundRecorder
- SoundSource() : sf::SoundSource
- SoundStream() : sf::SoundStream
- Space : sf::Keyboard
- Sprite() : sf::Sprite
- SrcAlpha : sf::BlendMode
- SrcColor : sf::BlendMode
- sRgbCapable : sf::ContextSettings
- start() : sf::SoundRecorder
- Status : sf::Ftp::Response, sf::Http::Response, sf::Socket, sf::Sound
- stencilBits : sf::ContextSettings
- stop() : sf::Sound, sf::SoundRecorder, sf::SoundStream
- Stopped : sf::SoundSource
- StrikeThrough : sf::Text
- String() : sf::String
- Style : sf::Text
- substring() : sf::String
- Subtract : sf::BlendMode, sf::Keyboard
- system : sf::Event::KeyEvent
- SystemStatus : sf::Ftp::Response
- SystemType : sf::Ftp::Response
Here is a list of all documented class members with links to the class documentation:

- t -

- T : sf::Keyboard
- Tab : sf::Keyboard
- Tcp : sf::Socket
- TcpListener() : sf::TcpListener
- TcpSocket() : sf::TcpSocket
- tell() : sf::FileInputStream, sf::InputStream, sf::MemoryInputStream
- terminate() : sf::Thread
- texCoords : sf::Vertex
- text : sf::Event
- Text() : sf::Text
- TextEntered : sf::Event
- texture : sf::RenderStates
- Texture() : sf::Texture
- textureRect : sf::Glyph
- Thread() : sf::Thread
- ThreadLocal() : sf::ThreadLocal
- ThreadLocalPtr() : sf::ThreadLocalPtr<T>
- Tilde : sf::Keyboard
- Time() : sf::Time
- toAnsi() : sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- to AnsiString() : sf::String
- toInteger() : sf::Color, sf::IpAddress
- toLatin1() : sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- top : sf::Rect<T>
- toString() : sf::IpAddress
- touch : sf::Event
- TouchBegan : sf::Event
- TouchEnded : sf::Event
- TouchMoved : sf::Event
- toUtf16() : sf::String, sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- toUtf32() : sf::String, sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- toUtf8() : sf::String, sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- toWide() : sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- toWideString() : sf::String
- TransferAborted : sf::Ftp::Response
- TransferMode : sf::Ftp
- transform : sf::RenderStates
- Transform() : sf::Transform
- Transformable() : sf::Transformable
- transformPoint() : sf::Transform
- `transformRect() : sf::Transform`
- `TransientContextLock() : sf::GlResource::TransientContextLock`
- `translate() : sf::Transform`
- `Transparent : sf::Color`
- `type : sf::Event::SensorEvent, sf::Event`
- `Type : sf::Sensor, sf::Shader, sf::Socket`
Here is a list of all documented class members with links to the class documentation:

- **U** -

- **U** : sf::Joystick, sf::Keyboard
- **Udp** : sf::Socket
- **UdpSocket()** : sf::UdpSocket
- **Unauthorized** : sf::Http::Response
- **unbind()** : sf::UdpSocket
- **Underlined** : sf::Text
- **unicode** : sf::Event::TextEvent
- **Unknown** : sf::Keyboard
- **unlock()** : sf::Mutex
- **unregisterReader()** : sf::SoundFileFactory
- **unregisterWriter()** : sf::SoundFileFactory
- **Up** : sf::Keyboard
- **update()** : sf::Joystick, sf::Shape, sf::Texture
- **upload()** : sf::Ftp
- **UserAcceleration**: `sf::Sensor`

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
Here is a list of all documented class members with links to the class documentation:

- **V** -

- **V :** sf::Joystick, sf::Keyboard
- **Vector2() :** sf::Vector2< T >
- **Vector3() :** sf::Vector3< T >
- **vendorId :** sf::Joystick::Identification
- **VersionNotSupported :** sf::Http::Response
- **Vertex :** sf::Shader, sf::Vertex
- **VertexArray() :** sf::VertexArray
- **VerticalWheel :** sf::Mouse
- **VideoMode() :** sf::VideoMode
- **View() :** sf::View
Here is a list of all documented class members with links to the class documentation:

### W

- **W** : sf::Keyboard
- **wait()** : sf::SocketSelector, sf::Thread
- **waitEvent()** : sf::Window
- **wheel** : sf::Event::MouseWheelScrollEvent
- **Wheel** : sf::Mouse
- **White** : sf::Color
- **width** : sf::Event::SizeEvent, sf::Rect< T >, sf::VideoMode
- **Window()** : sf::Window
- **write()** : sf::OutputSoundFile, sf::SoundFileWriter
Here is a list of all documented class members with links to the class documentation:

- X -

- x : sf::Event::MouseButtonEvent, sf::Event::MouseMoveEvent, sf::Event::MouseWheelEvent, sf::Event::MouseWheelScrollEvent, sf::Event::SensorEvent, sf::Event::TouchEvent

- X : sf::Joystick, sf::Keyboard

- x : sf::Vector2<T>, sf::Vector3<T>

- XButton1 : sf::Mouse

- XButton2 : sf::Mouse
Here is a list of all documented class members with links to the class documentation:

- y -

- y : sf::Event::MouseButtonEvent, sf::Event::MouseMoveEvent, sf::Event::MouseWheelEvent,
  sf::Event::MouseWheelScrollEvent, sf::Event::SensorEvent, sf::Event::

- Y : sf::Joystick, sf::Keyboard

- y : sf::Vector2< T >, sf::Vector3< T >

- Yellow : sf::Color
Here is a list of all documented class members with links to the class documentation:

- **Z -**

- `z : sf::Event::SensorEvent`
- `Z : sf::Joystick , sf::Keyboard`
- `z : sf::Vector3< T >`
- `Zero : sf::BlendMode , sf::Time`
- `zoom() : sf::View`
Here is a list of all documented class members with links to the class documentation:

- ~AlResource() : sf::AlResource
- ~Context() : sf::Context
- ~Drawable() : sf::Drawable
- ~FileInputStream() : sf::FileInputStream
- ~Font() : sf::Font
- ~Ftp() : sf::Ftp
- ~GlResource() : sf::GlResource
- ~Image() : sf::Image
- ~InputSoundFile() : sf::InputSoundFile
- ~InputStream() : sf::InputStream
- ~Lock() : sf::Lock
- ~Music() : sf::Music
- ~Mutex() : sf::Mutex
- ~OutputSoundFile() : sf::OutputSoundFile
- Packet() : sf::Packet
- RenderTarget() : sf::RenderTarget
- RenderTexture() : sf::RenderTexture
- RenderWindow() : sf::RenderWindow
- Shader() : sf::Shader
- Shape() : sf::Shape
- Socket() : sf::Socket
- SocketSelector() : sf::SocketSelector
- Sound() : sf::Sound
- SoundBuffer() : sf::SoundBuffer
- SoundBufferRecorder() : sf::SoundBufferRecorder
- SoundFileReader() : sf::SoundFileReader
- SoundFileWriter() : sf::SoundFileWriter
- SoundRecorder() : sf::SoundRecorder
- SoundSource() : sf::SoundSource
- SoundStream() : sf::SoundStream
- Texture() : sf::Texture
- Thread() : sf::Thread
- ThreadLocal() : sf::ThreadLocal
- Transformable() : sf::Transformable
- TransientContextLock() : sf::GlResource::TransientContextLock
- Window() : sf::Window
- a -

- accept() : sf::TcpListener
- add() : sf::SocketSelector
- AlResource() : sf::AlResource
- append() : sf::Packet, sf::VertexArray
- asMicroseconds() : sf::Time
- asMilliseconds() : sf::Time
- asSeconds() : sf::Time
- b -

- `begin()` : `sf::String`
- `bind()` : `sf::Shader, sf::Texture, sf::UdpSocket`
- `BlendMode()` : `sf::BlendMode`
- C -

- capture() : sf::RenderWindow
- changeDirectory() : sf::Ftp
- CircleShape() : sf::CircleShape
- clear() : sf::Packet, sf::RenderTarget, sf::SocketSelector, sf::String,
- Clock() : sf::Clock
- close() : sf::Socket, sf::TcpListener, sf::Window
- Color() : sf::Color
- combine() : sf::Transform
- connect() : sf::Ftp, sf::TcpSocket
- contains() : sf::Rect< T >
- Context() : sf::Context
- ContextSettings() : sf::ContextSettings
- ConvexShape() : sf::ConvexShape
- copy() : sf::Image
- `copyToImage()` : `sf::Texture`
- `count()` : `sf::Utf<16>`, `sf::Utf<32>`, `sf::Utf<8>`
- `create()` : `sf::Image`, `sf::RenderTexture`, `sf::Socket`, `sf::Texture`, `sf::Window`
- `createDirectory()` : `sf::Ftp`
- `createMaskFromColor()` : `sf::Image`
- `createReaderFromFilename()` : `sf::SoundFileFactory`
- `createReaderFromMemory()` : `sf::SoundFileFactory`
- `createReaderFromStream()` : `sf::SoundFileFactory`
- `createWriterFromFilename()` : `sf::SoundFileFactory`
- d -

- `decode()` : sf::Utf< 16 > , sf::Utf< 32 > , sf::Utf< 8 >
- `decodeAnsi()` : sf::Utf< 32 >
- `decodeWide()` : sf::Utf< 32 >
- `deleteDirectory()` : sf::Ftp
- `deleteFile()` : sf::Ftp
- `DirectoryResponse()` : sf::Ftp::DirectoryResponse
- `disconnect()` : sf::Ftp , sf::TcpSocket
- `display()` : sf::RenderTexture , sf::Window
- `download()` : sf::Ftp
- `draw()` : sf::Drawable , sf::RenderTarget
- e -

- encode() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >
- encodeAnsi() : sf::Utf< 32 >
- encodeWide() : sf::Utf< 32 >
- end() : sf::String
- endOfPacket() : sf::Packet
- ensureGIContext() : sf::GlResource
- erase() : sf::String
- f -

- FileInputStream() : sf::FileInputStream
- find() : sf::String
- findCharacterPos() : sf::Text
- flipHorizontally() : sf::Image
- flipVertically() : sf::Image
- Font() : sf::Font
- fromAnsi() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >
- fromLatin1() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >
- fromUtf16() : sf::String
- fromUtf32() : sf::String
- fromUtf8() : sf::String
- fromWide() : sf::Utf< 16 >, sf::Utf< 32 >, sf::Utf< 8 >
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
### Class Members

<table>
<thead>
<tr>
<th>Class List</th>
<th>Class Index</th>
<th>Class Hierarchy</th>
<th>Class Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Functions</td>
<td>Variables</td>
<td>Typedefs</td>
</tr>
</tbody>
</table>

### Functions

- `generateMipmap()` : sf::RenderTexture, sf::Texture
- `getActiveContext()` : sf::Context
- `getAttenuation()` : sf::SoundSource
- `getAvailableDevices()` : sf::SoundRecorder
- `getAxisPosition()` : sf::Joystick
- `getBody()` : sf::Http::Response
- `getBounds()` : sf::VertexArray
- `getBuffer()` : sf::Sound, sf::SoundBufferRecorder
- `getButtonCount()` : sf::Joystick
- `getCenter()` : sf::View
- `getChannelCount()` : sf::InputSoundFile, sf::SoundBuffer, sf::SoundRecorder
- `getCharacterSize()` : sf::Text
- `getColor()` : sf::Sprite, sf::Text
- `getData()` : sf::Packet, sf::String
- `getDataSize() : sf::Packet`
- `getDefaultDevice() : sf::SoundRecorder`
- `getDefaultView() : sf::RenderTarget`
- `getDesktopMode() : sf::VideoMode`
- `getDevice() : sf::SoundRecorder`
- `getDirection() : sf::Listener`
- `getDirectory() : sf::Ftp::DirectoryResponse`
- `getDirectoryListing() : sf::Ftp`
- `getDuration() : sf::InputSoundFile, sf::Music, sf::SoundBuffer`
- `getElapsedTime() : sf::Clock`
- `getField() : sf::Http::Response`
- `getFillColor() : sf::Shape, sf::Text`
- `getFont() : sf::Text`
- `getFullscreenModes() : sf::VideoMode`
- `getFunction() : sf::Context`
- `getGlobalBounds() : sf::Shape, sf::Sprite, sf::Text`
- `getGlobalVolume() : sf::Listener`
- `getGlyph() : sf::Font`
- `getHandle() : sf::Socket`
- `getIdentification() : sf::Joystick`
- `getInfo() : sf::Font`
- `getInverse() : sf::Transform`
- `getInverseTransform() : sf::Transformable, sf::View`
- `getKerning() : sf::Font`
- `getLineSpacing() : sf::Font`
- `getListing() : sf::Ftp::ListingResponse`
- getLocalAddress(): sf::IpAddress
- getLocalBounds(): sf::Shape, sf::Sprite, sf::Text
- getLocalPort(): sf::TcpListener, sf::TcpSocket, sf::UdpSocket
- getLoop(): sf::Sound, sf::SoundStream
- getMajorHttpVersion(): sf::Http::Response
- getMatrix(): sf::Transform
- getMaximumSize(): sf::Texture
- getMessage(): sf::Ftp::Response
- getMinDistance(): sf::SoundSource
- getMinorHttpVersion(): sf::Http::Response
- getNativeHandle(): sf::Shader, sf::Texture
- getOrigin(): sf::Transformable
- getOutlineColor(): sf::Shape, sf::Text
- getOutlineThickness(): sf::Shape, sf::Text
- getPitch(): sf::SoundSource
- getPixel(): sf::Image
- getPixelsPtr(): sf::Image
- getPlayingOffset(): sf::Sound, sf::SoundStream
- getPoint(): sf::CircleShape, sf::ConvexShape, sf::RectangleShape
- getPointCount(): sf::CircleShape, sf::ConvexShape, sf::RectangleShape
- getPosition(): sf::Listener, sf::Mouse, sf::SoundSource, sf::Touch
- getPrimitiveType(): sf::VertexArray
- getPublicAddress(): sf::IpAddress
- getRadius(): sf::CircleShape
- getRemoteAddress(): sf::TcpSocket
- getRemotePort(): sf::TcpSocket
- `getRotation()` : `sf::Transformable, sf::View`
- `getSampleCount()` : `sf::InputSoundFile, sf::SoundBuffer`
- `getSampleRate()` : `sf::InputSoundFile, sf::SoundBuffer, sf::SoundRecorder`
- `getSamples()` : `sf::SoundBuffer`
- `setScale()` : `sf::Transformable`
- `getSettings()` : `sf::Context, sf::Window`
- `getSize()` : `sf::FileInputStream, sf::Image, sf::InputStream, sf::Memo, sf::RectangleShape, sf::RenderTarget, sf::RenderTexture, sf::Renderer`, `sf::View, sf::Window`
- `getStatus()` : `sf::Ftp::Response, sf::Http::Response, sf::Sound, sf::SoundSource`
- `getString()` : `sf::Text`
- `getStyle()` : `sf::Text`
- `getSystemHandle()` : `sf::Window`
- `getTexture()` : `sf::Font, sf::RenderTexture, sf::Shape, sf::Sprite`
- `getTextureRect()` : `sf::Shape, sf::Sprite`
- `getTransform()` : `sf::Transformable, sf::View`
- `getUnderlinePosition()` : `sf::Font`
- `getUnderlineThickness()` : `sf::Font`
- `getUpVector()` : `sf::Listener`
- `getValue()` : `sf::Sensor, sf::ThreadLocal`
- `getVertexCount()` : `sf::VertexArray`
- `getView()` : `sf::RenderTarget`
- `getViewPort()` : `sf::RenderTarget, sf::View`
- `getVolume()` : `sf::SoundSource`
- `getWorkingDirectory()` : `sf::Ftp`
- `GlResource()` : `sf::GlResource`
- `Glyph()` : `sf::Glyph`
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
- h -

- hasAxis() : sf::Joystick
- hasFocus() : sf::Window
- Http() : sf::Http

---

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
- i -

- Image() : sf::Image
- initialize() : sf::RenderTarget, sf::SoundStream
- InputSoundFile() : sf::InputSoundFile
- insert() : sf::String
- intersects() : sf::Rect< T >
- IpAddress() : sf::IpAddress
- isAvailable() : sf::Sensor, sf::Shader, sf::SoundRecorder
- isBlocking() : sf::Socket
- isButtonPressed() : sf::Joystick, sf::Mouse
- isConnected() : sf::Joystick
- isDown() : sf::Touch
- isEmpty() : sf::String
- isExtensionAvailable() : sf::Context
- isGeometryAvailable() : sf::Shader
- isKeyPressed() : sf::Keyboard
- isOk() : sf::Ftp::Response
- isOpen() : sf::Window
- isReady() : sf::SocketSelector
- isRelativeToListener() : sf::SoundSource
- isRepeated() : sf::RenderTexture , sf::Texture
- isSmooth() : sf::RenderTexture , sf::Texture
- isSrgb() : sf::Texture
- isValid() : sf::VideoMode

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
- k -

- keepAlive() : sf::Ftp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen.
- sf::Thread
- sf::TcpListener
- sf::Ftp::ListingResponse
- sf::Font, sf::Image, sf::Shader, sf::SoundBuffer
- sf::Texture
- sf::Font, sf::Image, sf::Shader, sf::SoundBuffer
- sf::SoundBuffer
- sf::Font, sf::Image, sf::Shader, sf::SoundBuffer
- sf::Lock
- sf::Mutex
- sf::Ftp
- m -

- mapCoordsToPixel() : sf::RenderTarget
- mapPixelToCoords() : sf::RenderTarget
- MemoryInputStream() : sf::MemoryInputStream
- microseconds() : sf::Time
- milliseconds() : sf::Time
- move() : sf::Transformable, sf::View
- Music() : sf::Music
- Mutex() : sf::Mutex
- n -

- next() : sf::Utf< 16 > , sf::Utf< 32 > , sf::Utf< 8 >
- NonCopyable() : sf::NonCopyable
- O -

- onCreate() : sf::RenderWindow, sf::Window
- onGetData() : sf::Music, sf::SoundStream
- onProcessSamples() : sf::SoundBufferRecorder, sf::SoundRecorder
- onReceive() : sf::Packet
- onResize() : sf::RenderWindow, sf::Window
- onSeek() : sf::Music, sf::SoundStream
- onSend() : sf::Packet
- onStart() : sf::SoundBufferRecorder, sf::SoundRecorder
- onStop() : sf::SoundBufferRecorder, sf::SoundRecorder
- open() : sf::FileInputStream, sf::MemoryInputStream, sf::SoundFileReader
- openForWriting() : sf::InputSoundFile
- openFromFile() : sf::InputSoundFile, sf::Music, sf::OutputSoundFile
- openFromMemory() : sf::InputSoundFile, sf::Music
- openFromStream() : sf::InputSoundFile, sf::Music
- operator BoolType() : sf::Packet
- operator std::string() : sf::String
- operator std::wstring() : sf::String
- operator T *() : sf::ThreadLocalPtr< T >
- operator!=() : sf::BlendMode, sf::Color, sf::Rect< T >, sf::String, sf::Vector3< T >, sf::VideoMode
- operator%() : sf::Time
- operator%=() : sf::Time
- operator*() : sf::Color, sf::ThreadLocalPtr< T >, sf::Time, sf::Transform, sf::Vector2< T >, sf::Vector3< T >
- operator*=( ) : sf::Color, sf::Time, sf::Transform, sf::Vector2< T >, sf::Vector3< T >
- operator+() : sf::Color, sf::String, sf::Time, sf::Vector2< T >, sf::Vector3< T >
- operator+=( ) : sf::Color, sf::String, sf::Time, sf::Vector2< T >, sf::Vector3< T >
- operator-() : sf::Color, sf::Time, sf::Vector2< T >, sf::Vector3< T >
- operator-=( ) : sf::Color, sf::Time, sf::Vector2< T >, sf::Vector3< T >
- operator->() : sf::ThreadLocalPtr< T >
- operator/() : sf::Time, sf::Vector2< T >, sf::Vector3< T >
- operator/=() : sf::Time, sf::Vector2< T >, sf::Vector3< T >
- operator<() : sf::String, sf::Time, sf::VideoMode
- operator<() : sf::Packet
- operator<=() : sf::String, sf::Time, sf::VideoMode
- operator=() : sf::Font, sf::SocketSelector, sf::Sound, sf::SoundBuffer, sf::Texture, sf::ThreadLocalPtr< T >
- operator==() : sf::BlendMode, sf::Color, sf::Rect< T >, sf::String, sf::Vector3< T >, sf::VideoMode
- operator>() : sf::String, sf::Time, sf::VideoMode
- operator>=() : sf::String, sf::Time, sf::VideoMode
- `operator>>()` : `sf::Packet`
- `operator[]()` : `sf::String`, `sf::VertexArray`
- `OutputSoundFile()` : `sf::OutputSoundFile`
- p -

- Packet() : sf::Packet
- parentDirectory() : sf::Ftp
- pause() : sf::Sound, sf::SoundStream
- play() : sf::Sound, sf::SoundStream
- pollEvent() : sf::Window
- popGLStates() : sf::RenderTarget
- pushGLStates() : sf::RenderTarget
- r -

- `read()` : sf::FileInputStream, sf::InputSoundFile, sf::InputStream, sf::SoundFileReader
- `receive()` : sf::TcpSocket, sf::UdpSocket
- `Rect()` : sf::Rect< T >
- `RectangleShape()` : sf::RectangleShape
- `registerReader()` : sf::SoundFileFactory
- `registerWriter()` : sf::SoundFileFactory
- `remove()` : sf::SocketSelector
- `renameFile()` : sf::Ftp
- `RenderStates()` : sf::RenderStates
- `RenderTarget()` : sf::RenderTarget
- `RenderTexture()` : sf::RenderTexture
- `RenderWindow()` : sf::RenderWindow
- `replace()` : sf::String
- `Request()` : `sf::Http::Request`
- `requestFocus()` : `sf::Window`
- `reset()` : `sf::View`
- `resetBuffer()` : `sf::Sound`
- `resetGLStates()` : `sf::RenderTarget`
- `resize()` : `sf::VertexArray`
- `Response()` : `sf::Ftp::Response`, `sf::Http::Response`
- `restart()` : `sf::Clock`
- `rotate()` : `sf::Transform`, `sf::Transformable`, `sf::View`
- S -

- `saveToFile()` : sf::Image, sf::SoundBuffer
- `scale()` : sf::Transform, sf::Transformable
- `seconds()` : sf::Time
- `seek()` : sf::FileInputStream, sf::InputSoundFile, sf::InputStream, sf::SoundFileReader
- `send()` : sf::TcpSocket, sf::UdpSocket
- `sendCommand()` : sf::Ftp
- `sendRequest()` : sf::Http
- `setActive()` : sf::Context, sf::RenderTexture, sf::Window
- `setAttenuation()` : sf::SoundSource
- `setBlocking()` : sf::Socket
- `setBody()` : sf::Http::Request
- `setBuffer()` : sf::Sound
- `setCenter()` : sf::View
- setChannelCount() : sf::SoundRecorder
- setCharacterSize() : sf::Text
- setColor() : sf::Sprite, sf::Text
- setDevice() : sf::SoundRecorder
- setDirection() : sf::Listener
- setEnabled() : sf::Sensor
- setField() : sf::Http::Request
- setFillColor() : sf::Shape, sf::Text
- setFont() : sf::Text
- setFramerateLimit() : sf::Window
- setGlobalVolume() : sf::Listener
- setHost() : sf::Http
- setHttpVersion() : sf::Http::Request
- setIcon() : sf::Window
- setJoystickThreshold() : sf::Window
- setKeyRepeatEnabled() : sf::Window
- setLoop() : sf::Sound, sf::SoundStream
- setMethod() : sf::Http::Request
- setMinDistance() : sf::SoundSource
- setMouseCursorGrabbed() : sf::Window
- setMouseCursorVisible() : sf::Window
- setOrigin() : sf::Transformable
- setOutlineColor() : sf::Shape, sf::Text
- setOutlineThickness() : sf::Shape, sf::Text
- setParameter() : sf::Shader
- setPitch() : sf::SoundSource
- `setPixel()`: `sf::Image`
- `setPlayingOffset()`: `sf::Sound`, `sf::SoundStream`
- `setPoint()`: `sf::ConvexShape`
- `setPointCount()`: `sf::CircleShape`, `sf::ConvexShape`
- `setPosition()`: `sf::Listener`, `sf::Mouse`, `sf::SoundSource`, `sf::Transformable`
- `setPrimitiveType()`: `sf::VertexArray`
- `setProcessingInterval()`: `sf::SoundRecorder`
- `setRadius()`: `sf::CircleShape`
- `setRelativeToListener()`: `sf::SoundSource`
- `setRepeated()`: `sf::RenderTexture`, `sf::Texture`
- `setRotation()`: `sf::Transformable`, `sf::View`
- `setScale()`: `sf::Transformable`
- `setSize()`: `sf::RectangleShape`, `sf::View`, `sf::Window`
- `setSmooth()`: `sf::RenderTexture`, `sf::Texture`
- `setSrgb()`: `sf::Texture`
- `setString()`: `sf::Text`
- `setStyle()`: `sf::Text`
- `setTexture()`: `sf::Shape`, `sf::Sprite`
- `setTextureRect()`: `sf::Shape`, `sf::Sprite`
- `setTitle()`: `sf::Window`
- `setUniform()`: `sf::Shader`
- `setUniformArray()`: `sf::Shader`
- `setUpVector()`: `sf::Listener`
- `setUri()`: `sf::Http::Request`
- `setValue()`: `sf::ThreadLocal`
- `setVerticalSyncEnabled()`: `sf::Window`
- `setView()` : `sf::RenderTarget`
- `setViewport()` : `sf::View`
- `setVirtualKeyboardVisible()` : `sf::Keyboard`
- `setVisible()` : `sf::Window`
- `setVolume()` : `sf::SoundSource`
- `Shader()` : `sf::Shader`
- `Shape()` : `sf::Shape`
- `Socket()` : `sf::Socket`
- `SocketSelector()` : `sf::SocketSelector`
- `Sound()` : `sf::Sound`
- `SoundBuffer()` : `sf::SoundBuffer`
- `SoundRecorder()` : `sf::SoundRecorder`
- `SoundSource()` : `sf::SoundSource`
- `SoundStream()` : `sf::SoundStream`
- `Sprite()` : `sf::Sprite`
- `start()` : `sf::SoundRecorder`
- `stop()` : `sf::Sound`, `sf::SoundRecorder`, `sf::SoundStream`
- `String()` : `sf::String`
- `substring()` : `sf::String`
- T -

- TcpListener() : sf::TcpListener
- TcpSocket() : sf::TcpSocket
- tell() : sf::FileInputStream, sf::InputStream, sf::MemoryInputStream
- terminate() : sf::Thread
- Text() : sf::Text
- Texture() : sf::Texture
- Thread() : sf::Thread
- ThreadLocal() : sf::ThreadLocal
- ThreadLocalPtr() : sf::ThreadLocalPtr<T>
- Time() : sf::Time
- toAnsi() : sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- toAnsiString() : sf::String
- toInteger() : sf::Color, sf::IpAddress
- toLatin1() : sf::Utf<16>, sf::Utf<32>, sf::Utf<8>
- `toString()` : sf::IpAddress  
- `toUtf16()` : sf::String, sf::Utf<16>, sf::Utf<32>, sf::Utf<8>  
- `toUtf32()` : sf::String, sf::Utf<16>, sf::Utf<32>, sf::Utf<8>  
- `toUtf8()` : sf::String, sf::Utf<16>, sf::Utf<32>, sf::Utf<8>  
- `toWide()` : sf::Utf<16>, sf::Utf<32>, sf::Utf<8>  
- `toWideString()` : sf::String  
- `Transform()` : sf::Transform  
- `Transformable()` : sf::Transformable  
- `transformPoint()` : sf::Transform  
- `transformRect()` : sf::Transform  
- `TransientContextLock()` : sf::GlResource::TransientContextLock  
- `translate()` : sf::Transform
- U -

- UdpSocket() : sf::UdpSocket
- unbind() : sf::UdpSocket
- unlock() : sf::Mutex
- unregisterReader() : sf::SoundFileFactory
- unregisterWriter() : sf::SoundFileFactory
- update() : sf::Joystick, sf::Shape, sf::Texture
- upload() : sf::Ftp
- V -

- Vector2() : sf::Vector2< T >
- Vector3() : sf::Vector3< T >
- Vertex() : sf::Vertex
- VertexArray() : sf::VertexArray
- VideoMode() : sf::VideoMode
- View() : sf::View
## W

- `wait()` : sf::SocketSelector, sf::Thread
- `waitEvent()` : sf::Window
- `Window()` : sf::Window
- `write()` : sf::OutputSoundFile, sf::SoundFileWriter
- Z -

- zoom() : sf::View

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
- ~ -

- ~AIResource() : sf::AIResource
- ~Context() : sf::Context
- ~Drawable() : sf::Drawable
- FileInputStream() : sf::FileInputStream
- ~Font() : sf::Font
- ~Ftp() : sf::Ftp
- ~GlResource() : sf::GlResource
- ~Image() : sf::Image
- ~InputSoundFile() : sf::InputSoundFile
- ~InputStream() : sf::InputStream
- ~Lock() : sf::Lock
- ~Music() : sf::Music
- ~Mutex() : sf::Mutex
- ~OutputSoundFile() : sf::OutputSoundFile
- ~Packet() : sf::Packet
- ~RenderTarget() : sf::RenderTarget
- ~RenderTexture() : sf::RenderTexture
- ~RenderWindow() : sf::RenderWindow
- ~Shader() : sf::Shader
- ~Shape() : sf::Shape
- ~Socket() : sf::Socket
- ~SocketSelector() : sf::SocketSelector
- ~Sound() : sf::Sound
- ~SoundBuffer() : sf::SoundBuffer
- ~SoundBufferRecorder() : sf::SoundBufferRecorder
- ~SoundFileReader() : sf::SoundFileReader
- ~SoundFileWriter() : sf::SoundFileWriter
- ~SoundRecorder() : sf::SoundRecorder
- ~SoundSource() : sf::SoundSource
- ~SoundStream() : sf::SoundStream
- ~Texture() : sf::Texture
- ~Thread() : sf::Thread
- ~ThreadLocal() : sf::ThreadLocal
- ~Transformable() : sf::Transformable
- ~TransientContextLock() : sf::GlResource::TransientContextLock
- ~Window() : sf::Window
- a -

- a : sf::Color
- advance : sf::Glyph
- alphaDstFactor : sf::BlendMode
- alphaEquation : sf::BlendMode
- alphaSrcFactor : sf::BlendMode
- alt : sf::Event::KeyEvent
- antialiasingLevel : sf::ContextSettings
- Any : sf::IpAddress
- attributeFlags : sf::ContextSettings
- axis : sf::Event::JoystickMoveEvent

- b -

- b : sf::Color
- C -

- d -

- f -
- g -

- h -

- i -

- j -

- k -

- l -
- l -
  - left : sf::Rect< T >
  - LocalHost : sf::IpAddress

- m -
  - m_source : sf::SoundSource
  - Magenta : sf::Color
  - majorVersion : sf::ContextSettings
  - minorVersion : sf::ContextSettings
  - mouseButton : sf::Event
  - mouseMove : sf::Event
  - mouseWheel : sf::Event
  - mouseWheelScroll : sf::Event

- n -
  - name : sf::Joystick::Identification
  - None : sf::IpAddress

- p -
  - position : sf::Event::JoystickMoveEvent , sf::Vertex
  - productId : sf::Joystick::Identification

- r -
  - r : sf::Color
- Red : sf::Color

- S -

- sampleCount : sf::SoundFileReader::Info , sf::SoundStream::Chunk
- sampleRate : sf::SoundFileReader::Info
- samples : sf::SoundStream::Chunk
- sensor : sf::Event
- shader : sf::RenderStates
- shift : sf::Event::KeyEvent
- size : sf::Event
- sRgbCapable : sf::ContextSettings
- stencilBits : sf::ContextSettings
- system : sf::Event::KeyEvent

- t -

- texCoords : sf::Vertex
- text : sf::Event
- texture : sf::RenderStates
- textureRect : sf::Glyph
- top : sf::Rect< T >
- touch : sf::Event
- transform : sf::RenderStates
- Transparent : sf::Color
- type : sf::Event::SensorEvent , sf::Event
- u -
  - unicode : sf::Event::TextEvent

- v -
  - vendorId : sf::Joystick::Identification

- w -
  - wheel : sf::Event::MouseWheelScrollEvent
  - White : sf::Color
  - width : sf::Event::SizeEvent, sf::Rect< T >, sf::VideoMode

- x -
  - x : sf::Event::MouseButtonEvent, sf::Event::MouseMoveEvent, sf::Event::MouseWheelScrollEvent, sf::Event::SensorEvent, sf::Event::TouchEvent, sf::Vector3< T >

- y -
  - y : sf::Event::MouseButtonEvent, sf::Event::MouseMoveEvent, sf::Event::MouseWheelScrollEvent, sf::Event::SensorEvent, sf::Event::TouchEvent, sf::Vector3< T >
  - Yellow : sf::Color

- z -
  - z : sf::Event::SensorEvent, sf::Vector3< T >
Zero : sf::Time
- ConstIterator : sf::String
- Iterator : sf::String

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doc
- **Attribute**: sf::ContextSettings
- **Axis**: sf::Joystick
- **Button**: sf::Mouse
- **CoordinateType**: sf::Texture
- **Equation**: sf::BlendMode
- **EventType**: sf::Event
- **Factor**: sf::BlendMode
- **Key**: sf::Keyboard
- **Method**: sf::Http::Request
- **Status**: sf::Ftp::Response, sf::Http::Response, sf::Socket, sf::Sound
- **Style**: sf::Text
- **TransferMode**: sf::Ftp
- **Type**: sf::Sensor, sf::Shader, sf::Socket
- **Wheel**: sf::Mouse
SFML 2.4.2

- a -

- A : sf::Keyboard
- Accelerometer : sf::Sensor
- Accepted : sf::Http::Response
- Add : sf::BlendMode, sf::Keyboard
- AnyPort : sf::Socket
- Ascii : sf::Ftp
- AxisCount : sf::Joystick
- b -

- B : sf::Keyboard
- BackSlash : sf::Keyboard
- BackSpace : sf::Keyboard
- BadCommandSequence : sf::Ftp::Response
- BadGateway : sf::Http::Response
- BadRequest : sf::Http::Response
- Binary : sf::Ftp
- Bold : sf::Text
- ButtonCount : sf::Joystick , sf::Mouse
- C -

- C : sf::Keyboard
- Closed : sf::Event
- ClosingConnection : sf::Ftp::Response
- ClosingDataConnection : sf::Ftp::Response
- Comma : sf::Keyboard
- CommandNotImplemented : sf::Ftp::Response
- CommandUnknown : sf::Ftp::Response
- ConnectionClosed : sf::Ftp::Response
- ConnectionFailed : sf::Ftp::Response, sf::Http::Response
- Core : sf::ContextSettings
- Count : sf::Event, sf::Joystick, sf::Sensor
- Created : sf::Http::Response
- d -

- D : sf::Keyboard
- Dash : sf::Keyboard
- DataConnectionAlreadyOpened : sf::Ftp::Response
- DataConnectionOpened : sf::Ftp::Response
- DataConnectionUnavailable : sf::Ftp::Response
- Debug : sf::ContextSettings
- Default : sf::ContextSettings
- Delete : sf::Http::Request, sf::Keyboard
- DirectoryOk : sf::Ftp::Response
- DirectoryStatus : sf::Ftp::Response
- Disconnected : sf::Socket
- Divide : sf::Keyboard
- Done : sf::Socket
- Down : sf::Keyboard
- **DstAlpha**: `sf::BlendMode`
- **DstColor**: `sf::BlendMode`
- e -

- E : sf::Keyboard
- Ebcdic : sf::Ftp
- End : sf::Keyboard
- EnteringPassiveMode : sf::Ftp::Response
- Equal : sf::Keyboard
- Error : sf::Socket
- Escape : sf::Keyboard

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
- f -

- F : sf::Keyboard
- F1 : sf::Keyboard
- F10 : sf::Keyboard
- F11 : sf::Keyboard
- F12 : sf::Keyboard
- F13 : sf::Keyboard
- F14 : sf::Keyboard
- F15 : sf::Keyboard
- F2 : sf::Keyboard
- F3 : sf::Keyboard
- F4 : sf::Keyboard
- F5 : sf::Keyboard
- F6 : sf::Keyboard
- F7 : sf::Keyboard
- F8: sf::Keyboard
- F9: sf::Keyboard
- FileActionAborted: sf::Ftp::Response
- FileActionOk: sf::Ftp::Response
- FilenameNotAllowed: sf::Ftp::Response
- FileStatus: sf::Ftp::Response
- FileUnavailable: sf::Ftp::Response
- Forbidden: sf::Http::Response
- Fragment: sf::Shader

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
- g -

- G : sf::Keyboard
- GainedFocus : sf::Event
- GatewayTimeout : sf::Http::Response
- Geometry : sf::Shader
- Get : sf::Http::Request
- Gravity : sf::Sensor
- Gyroscope : sf::Sensor
- h -

- H : sf::Keyboard
- Head : sf::Http::Request
- HelpMessage : sf::Ftp::Response
- Home : sf::Keyboard
- HorizontalWheel : sf::Mouse
- i -

- I : sf::Keyboard
- Insert : sf::Keyboard
- InsufficientStorageSpace : sf::Ftp::Response
- InternalServerError : sf::Http::Response
- InvalidFile : sf::Ftp::Response
- InvalidResponse : sf::Ftp::Response , sf::Http::Response
- Italic : sf::Text

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
- j -

- J : sf::Keyboard
  - JoystickButtonPressed : sf::Event
  - JoystickButtonReleased : sf::Event
  - JoystickConnected : sf::Event
  - JoystickDisconnected : sf::Event
  - JoystickMoved : sf::Event

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen.
- k -

- K : sf::Keyboard
- KeyCount : sf::Keyboard
- KeyPressed : sf::Event
- KeyReleased : sf::Event
- l -

- L : sf::Keyboard
- LAlt : sf::Keyboard
- LBracket : sf::Keyboard
- LControl : sf::Keyboard
- Left : sf::Keyboard, sf::Mouse
- LocalError : sf::Ftp::Response
- LoggedIn : sf::Ftp::Response
- LostFocus : sf::Event
- LShift : sf::Keyboard
- LSystem : sf::Keyboard
<table>
<thead>
<tr>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>M : sf::Keyboard</td>
</tr>
<tr>
<td>Magnetometer : sf::Sensor</td>
</tr>
<tr>
<td>MaxDatagramSize : sf::UdpSocket</td>
</tr>
<tr>
<td>Menu : sf::Keyboard</td>
</tr>
<tr>
<td>Middle : sf::Mouse</td>
</tr>
<tr>
<td>MouseButtonPressed : sf::Event</td>
</tr>
<tr>
<td>MouseButtonReleased : sf::Event</td>
</tr>
<tr>
<td>MouseEntered : sf::Event</td>
</tr>
<tr>
<td>MouseLeft : sf::Event</td>
</tr>
<tr>
<td>MouseMoved : sf::Event</td>
</tr>
<tr>
<td>MouseWheelMoved : sf::Event</td>
</tr>
<tr>
<td>MouseWheelScrolled : sf::Event</td>
</tr>
<tr>
<td>MovedPermanently : sf::Http::Response</td>
</tr>
<tr>
<td>MovedTemporarily : sf::Http::Response</td>
</tr>
</tbody>
</table>
- **MultipleChoices**: `sf::Http::Response`
- **Multiply**: `sf::Keyboard`
- n -

- N : sf::Keyboard
- NeedAccountToLogIn : sf::Ftp::Response
- NeedAccountToStore : sf::Ftp::Response
- NeedInformation : sf::Ftp::Response
- NeedPassword : sf::Ftp::Response
- NoContent : sf::Http::Response
- Normalized : sf::Texture
- NotEnoughMemory : sf::Ftp::Response
- NotFound : sf::Http::Response
- NotImplemented : sf::Http::Response
- NotLoggedIn : sf::Ftp::Response
- NotModified : sf::Http::Response
- NotReady : sf::Socket
- Num0 : sf::Keyboard
- Num1 : sf::Keyboard
- Num2 : sf::Keyboard
- Num3 : sf::Keyboard
- Num4 : sf::Keyboard
- Num5 : sf::Keyboard
- Num6 : sf::Keyboard
- Num7 : sf::Keyboard
- Num8 : sf::Keyboard
- Num9 : sf::Keyboard
- Numpad0 : sf::Keyboard
- Numpad1 : sf::Keyboard
- Numpad2 : sf::Keyboard
- Numpad3 : sf::Keyboard
- Numpad4 : sf::Keyboard
- Numpad5 : sf::Keyboard
- Numpad6 : sf::Keyboard
- Numpad7 : sf::Keyboard
- Numpad8 : sf::Keyboard
- Numpad9 : sf::Keyboard

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
- O -

- O : sf::Keyboard
- Ok : sf::Ftp::Response, sf::Http::Response
- One : sf::BlendMode
- OneMinusDstAlpha : sf::BlendMode
- OneMinusDstColor : sf::BlendMode
- OneMinusSrcAlpha : sf::BlendMode
- OneMinusSrcColor : sf::BlendMode
- OpeningDataConnection : sf::Ftp::Response
- Orientation : sf::Sensor
- p -

- P : sf::Keyboard
- PageDown : sf::Keyboard
- PageTypeUnknown : sf::Ftp::Response
- PageUp : sf::Keyboard
- ParameterNotImplemented : sf::Ftp::Response
- ParametersUnknown : sf::Ftp::Response
- Partial : sf::Socket
- PartialContent : sf::Http::Response
- Pause : sf::Keyboard
- Paused : sf::SoundSource
- Period : sf::Keyboard
- Pixels : sf::Texture
- Playing : sf::SoundSource
- PointlessCommand : sf::Ftp::Response
- Post : sf::Http::Request
- PovX : sf::Joystick
- PovY : sf::Joystick
- Put : sf::Http::Request
- q -

- Q : sf::Keyboard
- Quote : sf::Keyboard
### SFML 2.4.2

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class List</td>
<td>Class Index</td>
<td>Class Hierarchy</td>
<td>Class Members</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>Functions</td>
<td>Variables</td>
<td>Typedefs</td>
<td>Enumerations</td>
<td>Enumerator</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
</tbody>
</table>

- **R**:
  - R : sf::Joystick , sf::Keyboard
  - RAlt : sf::Keyboard
  - RangeNotSatisfiable : sf::Http::Response
  - RBracket : sf::Keyboard
  - RControl : sf::Keyboard
  - Regular : sf::Text
  - ResetContent : sf::Http::Response
  - Resized : sf::Event
  - RestartMarkerReply : sf::Ftp::Response
  - Return : sf::Keyboard
  - ReverseSubtract : sf::BlendMode
  - Right : sf::Keyboard , sf::Mouse
  - RShift : sf::Keyboard
  - RSystem : sf::Keyboard
- S -

- S : sf::Keyboard
- SemiColon : sf::Keyboard
- SensorChanged : sf::Event
- ServiceNotAvailable : sf::Http::Response
- ServiceReady : sf::Ftp::Response
- ServiceReadySoon : sf::Ftp::Response
- ServiceUnavailable : sf::Ftp::Response
- Slash : sf::Keyboard
- Space : sf::Keyboard
- SrcAlpha : sf::BlendMode
- SrcColor : sf::BlendMode
- Stopped : sf::SoundSource
- StrikeThrough : sf::Text
- Subtract : sf::BlendMode, sf::Keyboard
- SystemStatus : sf::Ftp::Response
- SystemType : sf::Ftp::Response

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. Documentation generated by doxygen.
- t -

- T : sf::Keyboard
- Tab : sf::Keyboard
- Tcp : sf::Socket
- TextEntered : sf::Event
- Tilde : sf::Keyboard
- TouchBegan : sf::Event
- TouchEnded : sf::Event
- TouchMoved : sf::Event
- TransferAborted : sf:Ftp::Response
### Class Members

**- U -**

- U : sf::Joystick, sf::Keyboard
- Udp : sf::Socket
- Unauthorized : sf::Http::Response
- Underlined : sf::Text
- Unknown : sf::Keyboard
- Up : sf::Keyboard
- UserAcceleration : sf::Sensor

---

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
- V -

- **V** : sf::Joystick, sf::Keyboard
- **VersionNotSupported** : sf::Http::Response
- **Vertex** : sf::Shader
- **VerticalWheel** : sf::Mouse
- W -

- W : sf::Keyboard
- X -

- X : sf::Joystick, sf::Keyboard
- XButton1 : sf::Mouse
- XButton2 : sf::Mouse
- y -

- Y : sf::Joystick, sf::Keyboard
- Z -

- Z : sf::Joystick, sf::Keyboard
- Zero : sf::BlendMode
- `operator<` : `sf::IpAddress`
Here is a list of all documented files with brief descriptions:

- AlResource.hpp
- Audio.hpp
- BlendMode.hpp
- CircleShape.hpp
- Clock.hpp
- Color.hpp
- Config.hpp
- Context.hpp
- ContextSettings.hpp
- ConvexShape.hpp
- Drawable.hpp
- Err.hpp
- Event.hpp
- Audio/Export.hpp
- Graphics/Export.hpp
NonCopyable.hpp
OpenGL.hpp
OutputSoundFile.hpp
Packet.hpp
PrimitiveType.hpp
Rect.hpp
RectangleShape.hpp
RenderStates.hpp
RenderTarget.hpp
RenderTexture.hpp
RenderWindow.hpp
Sensor.hpp
Shader.hpp
Shape.hpp
Sleep.hpp
Socket.hpp
SocketHandle.hpp
SocketSelector.hpp
Sound.hpp
SoundBuffer.hpp
SoundBufferRecorder.hpp
SoundFileFactory.hpp
SoundFileReader.hpp
SoundFileWriter.hpp
SoundRecorder.hpp
SoundSource.hpp
SoundStream.hpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source code.

#ifndef SFML_ALRESOURCE_HPP
#define SFML_ALRESOURCE_HPP

namespace sf {

#include <SFML/Audio/Export.hpp>

namespace sf {

class SFML_AUDIO_API AlResource
{
    protected:
        AlResource();
        ~AlResource();
};

} // namespace sf

#endif // SFML_ALRESOURCE_HPP
Audio.hpp

1  //
2  // SFML - Simple and Fast Multimedia Library
3  // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4  //
5  // This software is provided 'as-is', without any express or implied warranty.
6  // In no event will the authors be held liable for any damages arising from the use
7  // of this software.
8  //
9  // Permission is granted to anyone to use this software for any purpose, including
10  // commercial applications and to alter it and redistribute it freely, subject to the
11  // following restrictions:
12  //
13  // 1. The origin of this software must not be misrepresented;
14  //    you must not claim that you wrote the original software.
15  //    If you use this software in a product, an acknowledgment in the product
16  //    documentation would be appreciated but is not required.
17  //
18  // 2. Altered source versions must be plainly marked as such, and must not be
19  //    misrepresented as being the original software.
20  //
21  // 3. This notice may not be removed or altered from any source.
22  //
23  //
24  #ifndef SFML_AUDIO_HPP
25  #define SFML_AUDIO_HPP
26  
27  // Headers
28  #include <SFML/System.hpp>
29  #include <SFML/Audio/InputSoundFile.hpp>
30  #include <SFML/Audio/Listener.hpp>
31  #include <SFML/Audio/Music.hpp>
#include <SFML/Audio/OutputSoundFile.hpp>
#include <SFML/Audio/Sound.hpp>
#include <SFML/Audio/SoundBuffer.hpp>
#include <SFML/Audio/SoundBufferRecorder.hpp>
#include <SFML/Audio/SoundFileFactory.hpp>
#include <SFML/Audio/SoundFileReader.hpp>
#include <SFML/Audio/SoundFileWriter.hpp>
#include <SFML/Audio/SoundRecorder.hpp>
#include <SFML/Audio/SoundSource.hpp>
#include <SFML/Audio/SoundStream.hpp>

#endif // SFML_AUDIO_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
BlendMode.hpp

1 // SFML - Simple and Fast Multimedia Library
2 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
3 //
4 // This software is provided 'as-is', without any express or implied warranty.
5 // In no event will the authors be held liable for any damages arising from the use of this software.
6 //
7 // Permission is granted to anyone to use this software for any purpose,
8 // including commercial applications, and to alter it and redistribute it freely,
9 // subject to the following restrictions:
10 //
11 // 1. The origin of this software must not be misrepresented;
12 //    you must not claim that you wrote the original software.
13 //    If you use this software in a product, an acknowledgment
14 //    in the product documentation would be appreciated but is not
15 //    required.
16 //
17 // 2. Altered source versions must be plainly marked as such, and
18 //    must not be misrepresented as being the original software.
19 //
20 // 3. This notice may not be removed or altered from any source
21 //
22 //
23 ifndef SFML_BLENDMODE_HPP
24 #define SFML_BLENDMODE_HPP
25 #ifdef SFML_BLENDMODE_HPP
26 #define SFML_BLENDMODE_HPP
27 #endif
28 // Headers
29 #include <SFML/Graphics/Export.hpp>
30
31 namespace sf
32 {
33 }
struct SFML_GRAPHICS_API BlendMode
{
    enum Factor
    {
        Zero,
        One,
        SrcColor,
        OneMinusSrcColor,
        DstColor,
        OneMinusDstColor,
        SrcAlpha,
        OneMinusSrcAlpha,
        DstAlpha,
        OneMinusDstAlpha
    }

    enum Equation
    {
        Add,
        Subtract,
        ReverseSubtract
    }

    BlendMode();
    BlendMode(Factor sourceFactor, Factor destinationFactor, Equation blendEquation = Add);
    BlendMode(Factor colorSourceFactor, Factor colorDestinationFactor, Equation colorBlendEquation, Factor alphaSourceFactor, Factor alphaDestinationFactor, Equation alphaBlendEquation);

    // Member Data
    Factor colorSrcFactor;
    Factor colorDstFactor;
    Equation colorEquation;
    Factor alphaSrcFactor;
    Factor alphaDstFactor;
    Equation alphaEquation;

    SFML_GRAPHICS_API bool operator == (const BlendMode& left, const BlendMode& right);
    SFML_GRAPHICS_API bool operator != (const BlendMode& left, const BlendMode& right);

    // Commonly used blending modes
    SFML_GRAPHICS_API extern const BlendMode BlendAlpha;
    SFML_GRAPHICS_API extern const BlendMode BlendAdd;
    SFML_GRAPHICS_API extern const BlendMode BlendMultiply;
    SFML_GRAPHICS_API extern const BlendMode BlendNone;
} // namespace sf
SFML is licensed under the terms and conditions of the [zlib/png](http://www.openalleg.com/)

Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
//
// 2. Modified source versions must be plainly marked as such, and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_CIRCLESHAPE_HPP
#define SFML_CIRCLESHAPE_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Shape.hpp>

namespace sf {
class SFML_GRAPHICS_API CircleShape : public Shape
{
    public:
    
exlicit CircleShape(float radius = 0, std::size_t pointCount = 30);
    void setRadius(float radius);
    float getRadius() const;
    void setPointCount(std::size_t count);
    virtual std::size_t getPointCount() const;
    virtual Vector2f getPoint(std::size_t index) const;

    private:
    
exlicit CircleShape(float radius = 0, std::size_t pointCount = 30);
    void setRadius(float radius);
    float getRadius() const;
    void setPointCount(std::size_t count);
    virtual std::size_t getPointCount() const;
    virtual Vector2f getPoint(std::size_t index) const;

    private:

    // Member data
    float m_radius;
    std::size_t m_pointCount;

}; // namespace sf

#endif // SFML_CIRCLESHPAE_HPP

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
Clock.hpp

```cpp
#ifndef SFML_CLOCK_HPP
#define SFML_CLOCK_HPP

namespace sf
```
class SFML_SYSTEM_API Clock {
  public:
  Clock();
  Time getElapsedTime() const;
  Time restart();

  private:
  // Member data
  Time m_startTime;
};

} // namespace sf

#include "SFML/Clock.hpp"

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_COLOR_HPP
#define SFML_COLOR_HPP

// Headers
#include <SFML/Graphics/Export.hpp>

namespace sf {


class SFML_GRAPHICS_API Color {
    public:
    Color();
    Color(Uint8 red, Uint8 green, Uint8 blue, Uint8 alpha = 255);
    explicit Color(Uint32 color);
    Uint32 toInteger() const;

    // Static member data
    static const Color Black;
    static const Color White;
    static const Color Red;
    static const Color Green;
    static const Color Blue;
    static const Color Yellow;
    static const Color Magenta;
    static const Color Cyan;
    static const Color Transparent;

    // Member data
    Uint8 r;
    Uint8 g;
    Uint8 b;
    Uint8 a;
};

SFML_GRAPHICS_API bool operator == (const Color& left, const Color& right);
SFML_GRAPHICS_API bool operator != (const Color& left, const Color& right);
SFML_GRAPHICS_API Color operator + (const Color& left, const Color& right);
SFML_GRAPHICS_API Color operator - (const Color& left, const Color& right);
SFML_GRAPHICS_API Color operator * (const Color& left, const Color& right);
SFML_GRAPHICS_API Color& operator +=(Color& left, const Color& right);
SFML_GRAPHICS_API Color& operator -= (Color& left, const Color& right);
SFML_GRAPHICS_API Color& operator *= (Color& left, const Color& right);

} // namespace sf

#endif // SFML_COLOR_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications,
// and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_CONFIG_HPP
#define SFML_CONFIG_HPP

#define SFML_VERSION_MAJOR 2
#define SFML_VERSION_MINOR 4
#define SFML_VERSION_PATCH 2
#endif
// Identify the operating system
// see
http://nadeausoftware.com/articles/2012/01/c_c_tip_how_use_compiler_predefined_macros_detect_operating_system

#if defined(_WIN32)
  // Windows
  #define SFML_SYSTEM_WINDOWS
  #ifndef NOMINMAX
    #define NOMINMAX
  #endif
#endif

#elif defined(__APPLE__) && defined(__MACH__)
  // Apple platform, see which one it is
  #include "TargetConditionals.h"

  #if TARGET_OS_IPHONE || TARGET_IPHONE_SIMULATOR
    // iOS
    #define SFML_SYSTEM_IOS
  #elif TARGET_OS_MAC
    // MacOS
    #define SFML_SYSTEM_MACOS
  #else
    // Unsupported Apple system
    #error This Apple operating system is not supported by SFML
  #endif

#elif defined(__unix__)
  // UNIX system, see which one it is
  #if defined(__ANDROID__)
    // Android
    #define SFML_SYSTEM_ANDROID
  #elif defined(__linux__) || defined(__FreeBSD__) || defined(__FreeBSD_kernel__)
    // Linux
    #define SFML_SYSTEM_LINUX
  #else
    // FreeBSD
    #define SFML_SYSTEM_FREEBSD
  #endif

#else
  // Unsupported platform
  #error This operating system is not supported by SFML
#endif
#else

// Unsupported UNIX system
#error This UNIX operating system is not supported by SFML library
#endif
#endif

// Define a portable debug macro
#if !defined(NDEBUG)
#define SFML_DEBUG
#endif

// Define helpers to create portable import / export macros for each module
#if !defined(SFML_STATIC)
#if defined(SFML_SYSTEM_WINDOWS)
// Windows compilers need specific (and different) keywords
#define SFML_API_EXPORT __declspec(dllexport)
#define SFML_API_IMPORT __declspec(dllimport)

// For Visual C++ compilers, we also need to turn off this
#ifdef _MSC_VER
#pragma warning(disable: 4251)
#endif
#endif
#else // Linux, FreeBSD, Mac OS X
#if __GNUC__ >= 4
// GCC 4 has special keywords for showing/hidding symbols
// the same keyword is used for both importing and exporting
#define SFML_API_EXPORT __attribute__(__(visibility
#define SFML_API_IMPORT __attribute__(__(visibility
#endif
#else
// GCC < 4 has no mechanism to explicitely hide symbols
#endif

#else

// Unsupported system
#error This operating system is not supported by SFML library
#endif
# SFML_API_EXPORT
#define SFML_API_IMPORT

#endif
#endif
#if defined(SFML_NO_DEPRECATED_WARNINGS)
#define SFML_DEPRECATED
#else
// Static build doesn't need import/export macros
#define SFML_API_EXPORT
#define SFML_API_IMPORT
#endif

// Cross-platform warning for deprecated functions and classes
// Usage:
// class SFML_DEPRECATED MyClass
// {
//    SFML_DEPRECATED void memberFunc();
// }
//
// SFML_DEPRECATED void globalFunc();
#if defined(SFML_NO_DEPRECATED_WARNINGS)
    // User explicitly requests to disable deprecation warnings
    #define SFML_DEPRECATED
#else
    // Other compilers are not supported, leave class or function as-is.
    #pragma message("SFML_DEPRECATED is not supported for your compiler, please contact the SFML team")
#endif
#define SFML_DEPRECATED
// Define portable fixed-size types
namespace sf
{
    // All "common" platforms use the same size for char, short
    // (basically there are 3 types for 3 sizes, so no other match
    // we can use them without doing any kind of check

    // 8 bits integer types
    typedef signed char Int8;
    typedef unsigned char Uint8;

    // 16 bits integer types
    typedef signed short Int16;
    typedef unsigned short Uint16;

    // 32 bits integer types
    typedef signed int Int32;
    typedef unsigned int Uint32;

    // 64 bits integer types
    #if defined(_MSC_VER)
        typedef signed __int64 Int64;
        typedef unsigned __int64 Uint64;
    #else
        typedef signed long long Int64;
        typedef unsigned long long Uint64;
    #endif

} // namespace sf

#endif // SFML_CONFIG_HPP
```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_CONTEXT_HPP
#define SFML_CONTEXT_HPP

// Headers
#include <SFML/Window/Export.hpp>
#include <SFML/Window/GlResource.hpp>
#include <SFML/Window/ContextSettings.hpp>
#include <SFML/System/NonCopyable.hpp>
```

---

**Context.hpp**

This file contains the header for the SFML Context class, which is part of the SFML library. SFML (Simple and Fast Multimedia Library) is a cross-platform multimedia library that provides classes for handling audio, graphics, and networking.

The code snippet above includes several include directives for headers that are necessary for the SFML library. These headers define various classes and functions that handle multimedia operations.

The library is distributed under the 'as-is' condition without any warranty. It is available for use in any purpose, including commercial applications, with some restrictions on its redistribution and modification.

SFML is under the copyright of Laurent Gomila, and the code is intended to be used and distributed freely as long as the original notice and attribution are maintained.

---

**License Agreements and Terms:**

SFML is released under the LGPL v2.1 license. For more information about the license, visit the SFML website or read the license text thoroughly. SFML is a free software library that aims to be simple and fast, suitable for a wide range of multimedia applications.

---

**Support and Contributions:**

SFML is maintained by its developers, and contributions are always welcome. If you encounter issues or have ideas for improvements, you can reach out to the development team through the provided contact details.

---

**Documentation and Resources:**

SFML provides comprehensive documentation and resources for developers. The documentation is available on the website, which includes tutorials, API documentation, and examples to help users understand and use the library effectively.

---

**Community and Feedback:**

The SFML community is active and friendly. Developers are encouraged to join the forums, attend meetups, and participate in discussions to share knowledge, ask questions, and contribute to the development of the library.

---

**Contributors:**

Laurent Gomila is the main contributor to SFML, but the project relies on the contributions of many developers. If you have contributed to the library, consider becoming a formal contributor or supporting the development team through donations.

---

**Glossary:**

- **SFML**: SFML is a multimedia library that provides classes for handling audio, graphics, and networking.
- **LGPL v2.1**: A copyleft license that allows users to use, modify, and distribute the library, but with the condition that any derivative works must also be under the LGPL v2.1.
- **Cross-platform**: SFML is designed to work on multiple operating systems, including Windows, macOS, and Linux.
- **Open source**: SFML is open source software, meaning its source code is available for anyone to view, modify, and distribute.

---

**Contact:**

For more information or to contribute, visit the SFML website or reach out to the development team through the provided contact details.
namespace sf
{
    namespace priv
    {
        class GlContext;
    }

typedef void (*GlFunctionPointer)();

class SFML_WINDOW_API Context : GlResource, NonCopyable
{
public:
    Context();
    ~Context();
    bool setActive(bool active);
    const ContextSettings& getSettings() const;
    static bool isExtensionAvailable(const char* name);
    static GlFunctionPointer getFunction(const char* name);
    static const Context* getActiveContext();
    Context(const ContextSettings& settings, unsigned int width,
    private:
        // Member data
        priv::GlContext* m_context;
    };
}

#endif // SFML_CONTEXT_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.
//
#ifndef SFML_CONTEXTSETTINGS_HPP
#define SFML_CONTEXTSETTINGS_HPP

namespace sf
{

struct ContextSettings
{
    enum Attribute

```cpp
{
    Default = 0,
    Core   = 1 << 0,
    Debug  = 1 << 2
};

explicit ContextSettings(unsigned int depth = 0,
                           unsigned int antialiasing = 0,
                           unsigned int major = 1,
                           unsigned int minor = 1,
                           Default, bool sRgb = false) :
    depthBits (depth),
    stencilBits (stencil),
    antialiasingLevel(antialiasing),
    majorVersion (major),
    minorVersion (minor),
    attributeFlags (attributes),
    sRgbCapable (sRgb)
{
    }

    // Member data
    unsigned int depthBits;
    unsigned int stencilBits;
    unsigned int antialiasingLevel;
    unsigned int majorVersion;
    unsigned int minorVersion;
    Uint32 attributeFlags;
    bool sRgbCapable;
};

} // namespace sf

#endif // SFML_CONTEXTSETTINGS_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
```
ConvexShape.hpp

// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment
//    in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_CONVEXSHAPE_HPP
#define SFML_CONVEXSHAPE_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Shape.hpp>
#include <vector>

#endif // SFML_CONVEXSHAPE_HPP
namespace sf
{
class SFML_GRAPHICS_API ConvexShape : public Shape
{
public:
    explicit ConvexShape(std::size_t pointCount = 0);
    void setPointCount(std::size_t count);
    virtual std::size_t getPointCount() const;
    void setPoint(std::size_t index, const Vector2f& point);
    virtual Vector2f getPoint(std::size_t index) const;

private:
    // Member data
    std::vector<Vector2f> m_points;
};
} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Drawable.hpp

```
1 //
3 // SFML - Simple and Fast Multimedia Library
4 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
5 //
6 // This software is provided 'as-is', without any express or implied
7 // warranty.  In no event will the authors be held liable for any damages
8 // arising from the use of this software.
9 //
10 // Permission is granted to anyone to use this software for any
11 // purpose, including commercial applications, and to alter it and
12 // redistribute it freely, subject to the following restrictions:
13 //
14 // 1. The origin of this software must not be misrepresented;
15 //    you must not claim that you wrote the original software.
16 //    If you use this software in a product, an acknowledgment
17 //    in the product documentation would be appreciated but is
18 //    not required.
19 //
20 // 2. Altered source versions must be plainly marked as such, and
21 //    must not be misrepresented as being the original software.
22 //
23 // 3. This notice may not be removed or altered from any source
24 //
25 #ifndef SFML_DRAWABLE_HPP
26 #define SFML_DRAWABLE_HPP
27
29 // Headers
30 #include <SFML/Graphics/Export.hpp>
31 #include <SFML/Graphics/RenderStates.hpp>
32
34 namespace sf
35 ```
class RenderTarget;

class SFML_GRAPHICS_API Drawable
{
public:
    virtual ~Drawable() {}  
protected:
    friend class RenderTarget;
    virtual void draw(RenderTarget& target, RenderStates states);
} // namespace sf  

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Err.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty. In no event will the authors be held liable for any damages a
7 // user of this software may claim.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redist
11 // it, subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 // that you wrote the original software.
15 // 2. If you use this software in a product, an acknowledgment
16 // in the product documentation would be appreciated but is not
17 // required.
18 // 2. Altered source versions must be plainly marked as such,
19 // and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 // file.
23 //
24 #ifndef SFML_ERR_HPP
25 #define SFML_ERR_HPP
26
27 // Headers
28 #include <SFML/System/Export.hpp>
29 #include <ostream>
30
31 namespace sf{
36  {
37     SFML_SYSTEM_API std::ostream& err();
38     }
39  // namespace sf
40
41  #endif  // SFML_ERR_HPP
42 43
44
45 46  #endif  // SFML.ERR_HPP
47
48
Event.hpp

```cpp
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty.  In no event will the authors be held liable for any damages
7 // arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redist
11 // it, subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 //    If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 //    and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 //
23 #ifndef SFML_EVENT_HPP
24 #define SFML_EVENT_HPP
25
26 #define SFML_EVENT_HPP
27
28 // Headers
29 #include <SFML/Config.hpp>
30 #include <SFML/Window/Joystick.hpp>
31 #include <SFML/Window/Keyboard.hpp>
32 #include <SFML/Window/Mouse.hpp>
33 #include <SFML/Window/Sensor.hpp>
```
namespace sf {

class Event {

public:

  struct SizeEvent {
    unsigned int width;
    unsigned int height;
  };

  struct KeyEvent {
    Keyboard::Key code;
    bool alt;
    bool control;
    bool shift;
    bool system;
  };

  struct TextEvent {
    Uint32 unicode;
  };

  struct MouseMoveEvent {
    int x;
    int y;
  };

  struct MouseButtonEvent {
    Mouse::Button button;
    int x;
    int y;
  };

  struct MouseWheelEvent {
    int delta;
    int x;
    int y;
  };

  struct MouseWheelScrollEvent {
    Mouse::Wheel wheel;
    float delta;
  }
};
int x;
int y;

struct JoystickConnectEvent
{
    unsigned int joystickId;
};

struct JoystickMoveEvent
{
    unsigned int joystickId;
    Joystick::Axis axis;
    float position;
};

struct JoystickButtonEvent
{
    unsigned int joystickId;
    unsigned int button;
};

struct TouchEvent
{
    unsigned int finger;
    int x;
    int y;
};

struct SensorEvent
{
    Sensor::Type type;
    float x;
    float y;
    float z;
};

enum EventType
{
    Closed,
    Resized,
    LostFocus,
    GainedFocus,
    TextEntered,
    KeyPressed,
    KeyReleased,
    MouseWheelMoved,
    MouseWheelScrolled,
    MouseButtonPressed,
    MouseButtonReleased,
    MouseMoved,
    MouseEntered,
MouseLeft,
JoystickButtonPressed,
JoystickButtonReleased,
JoystickMoved,
JoystickConnected,
JoystickDisconnected,
TouchBegan,
TouchMoved,
TouchEnded,
SensorChanged,
Count
};

// Member data
EventType type;
union
{
    SizeEvent size;
    KeyEvent key;
    TextEvent text;
    MouseMoveEvent mouseMove;
    MouseButtonEvent mouseButton;
    MouseWheelEvent mouseWheel;
    MouseWheelScrollEvent mouseWheelScroll;
    JoystickMoveEvent joystickMove;
    JoystickButtonEvent joystickButton;
    JoystickConnectEvent joystickConnect;
    TouchEvent touch;
    SensorEvent sensor;
};
} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
/* SFML - Simple and Fast Multimedia Library */
/* Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org) */
/* This software is provided 'as-is', without any express or implied warranty. */
/* In no event will the authors be held liable for any damages arising from the use of this software. */
/* Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: */
/* 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. */
/* 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. */
/* 3. This notice may not be removed or altered from any source distribution. */

#ifndef SFML_AUDIO_EXPORT_HPP
#define SFML_AUDIO_EXPORT_HPP

#include <SFML/Config.hpp>

#if defined(SFML_AUDIO_EXPORTS)
#define SFML_AUDIO_API SFML_API_EXPORT

#else

#define SFML_AUDIO_API SFML_API_IMPORT

#endif

#endif // SFML_AUDIO_EXPORT_HPP

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
/**
 * SFML - Simple and Fast Multimedia Library
 * Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
 * This software is provided 'as-is', without any express or implied
 * warranty. In no event will the authors be held liable for any damages a
 * software.
 * Permission is granted to anyone to use this software for any
 * including commercial applications, and to alter it and redist
 * subject to the following restrictions:
 * 1. The origin of this software must not be misrepresented;
 * you must not claim that you wrote the original software.
 * If you use this software in a product, an acknowledgment
 * in the product documentation would be appreciated but is not
 * required.
 * 2. Altered source versions must be plainly marked as such,
 * and must not be misrepresented as being the original software.
 * 3. This notice may not be removed or altered from any source
 *
 * #ifndef SFML_GRAPHICS_EXPORT_HPP
 * #define SFML_GRAPHICS_EXPORT_HPP

 // Define portable import / export macros
 #if defined(SFML_GRAPHICS_EXPORTS)

 // Headers
 #include <SFML/Config.hpp>

 */
#define SFML_GRAPHICS_API SFML_API_EXPORT
#else
#define SFML_GRAPHICS_API SFML_API_IMPORT
#endif
#endif // SFML_GRAPHICS_EXPORT_HPP

SFML is licensed under the terms and conditions of the zlib/png licensing.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_NETWORK_EXPORT_HPP
#define SFML_NETWORK_EXPORT_HPP

// Define portable import / export macros
#if defined(SFML_NETWORK_EXPORTS)
#define SFML_NETWORK_API SFML_API_EXPORT

#else
#define SFML_NETWORK_API SFML_API_IMPORT
#endif

#endif // SFML_NETWORK_EXPORT_HPP

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source code.

#ifndef SFML_SYSTEM_EXPORT_HPP
#define SFML_SYSTEM_EXPORT_HPP

// Define portable import / export macros
#if defined(SFML_SYSTEM_EXPORT_HPP)
#define SFML_SYSTEM_EXPORT_HPP
#endif

// Headers
#include <SFML/Config.hpp>

//
#define SFML_SYSTEM_API SFML_API_EXPORT
#else
#define SFML_SYSTEM_API SFML_API_IMPORT
#endif
#endif // SFML_SYSTEM_EXPORT_HPP
```cpp
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty.  In no event will the authors be held liable for any damages a
7 // person may claim arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redist
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //     you must not claim that you wrote the original software.
15 //     If you use this software in a product, an acknowledgment
16 //     in the product documentation would be appreciated but is not
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 //     and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 //
23 #ifndef SFML_WINDOW_EXPORT_HPP
24 #define SFML_WINDOW_EXPORT_HPP
25 #if defined(SFML_WINDOW_EXPORTS)
26 #define SFML_WINDOW_EXPORT_HPP
27 // Headers
28 #include <SFML/Config.hpp>
29 // Define portable import / export macros
30 #if defined(SFML_WINDOW_EXPORTS)
```
```c
#define SFML_WINDOW_API SFML_API_EXPORT
#else
#define SFML_WINDOW_API SFML_API_IMPORT
#endif
#endif // SFML_WINDOW_EXPORT_HPP
```

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
SFML - Simple and Fast Multimedia Library
Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source.

#ifndef SFML_FILEINPUTSTREAM_HPP
#define SFML_FILEINPUTSTREAM_HPP

// Headers
#include <SFML/Config.hpp>
#include <SFML/System/Export.hpp>
#include <SFML/System/InputStream.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <cstdio>

#endif // SFML_FILEINPUTSTREAM_HPP
#include <string>

#ifdef ANDROID
namespace sf
{  
namespace priv
{  
class SFML_SYSTEM_API ResourceStream;
}
#endif

namespace sf
{
class SFML_SYSTEM_API FileInputStream : public InputStream, Non
{
public:
   FileInputStream();
   virtual ~FileInputStream();
   bool open(const std::string& filename);
   virtual Int64 read(void* data, Int64 size);
   virtual Int64 seek(Int64 position);
   virtual Int64 tell();
   virtual Int64 getSize();
private:
   // Member data
   #ifdef ANDROID
   priv::ResourceStream* m_file;
   #else
   std::FILE* m_file;
   #endif
};

} // namespace sf
#endif // SFML_FILEINPUTSTREAM_HPP
SFML - Simple and Fast Multimedia Library
Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

This software is provided 'as-is', without any express or implied warranty.
In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_FONT_HPP
#define SFML_FONT_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Glyph.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/System/Vector2.hpp>

#include <SFML/System/String.hpp>
#include <map>
#include <string>
#include <vector>

namespace sf {

class InputStream;

class SFML_GRAPHICS_API Font {
public:

  struct Info {
    std::string family;
  };

public:

  Font();
  Font(const Font& copy);
  ~Font();

  bool loadFromFile(const std::string& filename);
  bool loadFromMemory(const void* data, std::size_t sizeInBytes);
  bool loadFromStream(InputStream& stream);

  const Info& getInfo() const;

  const Glyph& getGlyph(Uint32 codePoint, unsigned int characterSize, outlineThickness = 0) const;

  float getKerning(Uint32 first, Uint32 second, unsigned int characterSize) const;

  float getLineSpacing(unsigned int characterSize) const;

  float getUnderlinePosition(unsigned int characterSize) const;

  float getUnderlineThickness(unsigned int characterSize) const;

  const Texture& getTexture(unsigned int characterSize) const;

  Font& operator =(const Font& right);

private:

```cpp
struct Row
{
    Row(unsigned int rowTop, unsigned int rowHeight)
    : width(0), height(rowHeight) {}
    unsigned int width;
    unsigned int top;
    unsigned int height;
};

// Types
typedef std::map<Uint64, Glyph> GlyphTable;

struct Page
{
    Page();
    GlyphTable glyphs;
    Texture texture;
    unsigned int nextRow;
    std::vector<Row> rows;
};

void cleanup();

Glyph loadGlyph(Uint32 codePoint, unsigned int characterSize,
                unsigned int outlineThickness) const;

IntRect findGlyphRect(Page& page, unsigned int width, unsigned
                       int height) const;

bool setCurrentSize(unsigned int characterSize) const;

// Types
typedef std::map<unsigned int, Page> PageTable;

// Member data
void* m_library;
void* m_face;
void* m_streamRec;
void* m_stroker;
int* m_refCount;
Info m_info;
mutable PageTable m_pages;
mutable std::vector<Uint8> m_pixelBuffer;
#ifdef SFML_SYSTEM_ANDROID
void* m_stream;
#endif

}; // namespace sf
```
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
SFML - Simple and Fast Multimedia Library
Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented;
   you must not claim that you wrote the original software.
2. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
3. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

This notice may not be removed or altered from any source distribution.

#ifndef SFML_FTP_HPP
#define SFML_FTP_HPP

// Headers
#include <SFML/Network/Export.hpp>
#include <SFML/Network/TcpSocket.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <SFML/System/Time.hpp>
#include <string>

#include <vector>

namespace sf {
    class IpAddress;
}

class SFML_NETWORK_API Ftp : NonCopyable {
public:
    enum TransferMode {
        Binary,
        Ascii,
        Ebcdic
    };
}

class SFML_NETWORK_API Response {
public:
    enum Status {
        // 1xx: the requested action is being initiated, expect another reply before proceeding with a new command
        RestartMarkerReply = 110,
        ServiceReadySoon = 120,
        DataConnectionAlreadyOpened = 125,
        OpeningDataConnection = 150,

        // 2xx: the requested action has been successfully completed
        Ok = 200,
        PointlessCommand = 202,
        SystemStatus = 211,
        DirectoryStatus = 212,
        FileStatus = 213,
        HelpMessage = 214,
        SystemType = 215,
        ServiceReady = 220,
        ClosingConnection = 221,
        DataConnectionOpened = 225,
        ClosingDataConnection = 226,
        EnteringPassiveMode = 227,
        LoggedIn = 230,
        FileActionOk = 250,
        DirectoryOk = 257,

        // 3xx: the command has been accepted, but the requested action is dormant, pending receipt of further information
        NeedPassword = 331,
        NeedAccountToLogIn = 332,
    }
}
NeedInformation = 350,

// 4xx: the command was not accepted and the requested action did not take place
// but the error condition is temporary and the action may be requested again
ServiceUnavailable = 421,
DataConnectionUnavailable = 425,
TransferAborted = 426,
FileActionAborted = 450,
LocalError = 451,
InsufficientStorageSpace = 452,

// 5xx: the command was not accepted and the requested command did not take place
CommandUnknown = 500,
ParametersUnknown = 501,
CommandNotImplemented = 502,
BadCommandSequence = 503,
ParameterNotImplemented = 504,
NotLoggedIn = 530,
NeedAccountToStore = 532,
FileUnavailable = 550,
PageTypeUnknown = 551,
NotEnoughMemory = 552,
FilenameNotAllowed = 553,

// 10xx: SFML custom codes
InvalidResponse = 1000,
ConnectionFailed = 1001,
ConnectionClosed = 1002,
InvalidFile = 1003

};

explicit Response(Status code = InvalidResponse, const std::string& message =);

bool isOk() const;

Status getStatus() const;

const std::string& getMessage() const;

private:

// Member data
Status m_status;
std::string m_message;

};

class SFML_NETWORK_API DirectoryResponse : public Response
{
public:

DirectoryResponse(const Response& response);
const std::string& getDirectory() const;

private:

// Member data
std::string m_directory;

};

class SFML_NETWORK_API ListingResponse : public Response
{
public:

ListingResponse(const Response& response, const std::string& data);

const std::vector<std::string>& getListing() const;

private:

// Member data
std::vector<std::string> m_listing;

};

~Ftp();

Response connect(const IpAddress& server, unsigned short port = 21, Time::Zero);

Response disconnect();

Response login();

Response login(const std::string& name, const std::string& password);

Response keepAlive();

DirectoryResponse getWorkingDirectory();

ListingResponse getDirectoryListing(const std::string& directory = Response);

Response changeDirectory(const std::string& directory);

Response parentDirectory();

Response createDirectory(const std::string& name);

Response deleteDirectory(const std::string& name);

Response renameFile(const std::string& file, const std::string& newName);
Response deleteFile(const std::string& name);
Response download(const std::string& remoteFile, const std::string& localFile, TransferMode mode = Binary);
Response upload(const std::string& localFile, const std::string& remotePath, mode = Binary);
Response sendCommand(const std::string& command, const std::string& parameter =
private:
Response getResponse();
class DataChannel;
friend class DataChannel;
// Member data
TcpSocket m_commandSocket;
std::string m_receiveBuffer;
};
#endif // SFML_FTP_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
GlResource.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty. In no event will the authors be held liable for any damages
7 // arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redist
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 //    If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 //    and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 //
23 #ifndef SFML_GLRESOURCE_HPP
24 #define SFML_GLRESOURCE_HPP
25
26 // Headers
27 #include <SFML/Window/Export.hpp>
28 #include <SFML/System/NonCopyable.hpp>
29
30 namespace sf {
class Context;

class SFML_WINDOW_API GlResource
{
protected:
    GlResource();
    ~GlResource();
    static void ensureGlContext();
}

class SFML_WINDOW_API TransientContextLock : NonCopyable
{
public:
    TransientContextLock();
    ~TransientContextLock();

private:
    Context* m_context;

};

} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied warranty.
6 // In no event will the authors be held liable for any damages arising from the use of this software.
7 //
8 // Permission is granted to anyone to use this software for any purpose:
9 // including commercial applications, and to alter it and redistribute it freely,
10 // subject to the following restrictions:
11 //
12 // 1. The origin of this software must not be misrepresented;
13 //    you must not claim that you wrote the original software.
14 //    If you use this software in a product, an acknowledgment
15 //    in the product documentation would be appreciated but is not
16 //    required.
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 //    and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 //
23 #ifndef SFML_GLSL_HPP
24 #define SFML_GLSL_HPP
25 // Headers
26 #include <SFML/Graphics/Transform.hpp>
27 #include <SFML/Graphics/Color.hpp>
28 #include <SFML/System/Vector2.hpp>
29 #include <SFML/System/Vector3.hpp>
namespace sf
{
namespace priv
{
    // Forward declarations
    template <std::size_t Columns, std::size_t Rows>
    struct Matrix;

    template <typename T>
    struct Vector4;

    #include <SFML/Graphics/Glsl.inl>

} // namespace priv

namespace Glsl
{
    typedef Vector2<float> Vec2;
    typedef Vector2<int> Ivec2;
    typedef Vector2<bool> Bvec2;
    typedef Vector3<float> Vec3;
    typedef Vector3<int> Ivec3;
    typedef Vector3<bool> Bvec3;

    #ifdef SFML_DOXYGEN
    typedef implementation-defined Vec4;
    typedef implementation-defined Ivec4;
    typedef implementation-defined Bvec4;
    typedef implementation-defined Mat3;
    typedef implementation-defined Mat4;
    #else // SFML_DOXYGEN
    typedef priv::Vector4<float> Vec4;
    typedef priv::Vector4<int> Ivec4;
    typedef priv::Vector4<bool> Bvec4;
    typedef priv::Matrix<3, 3> Mat3;
    typedef priv::Matrix<4, 4> Mat4;
    #endif
} // namespace Glsl
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by Doxygen.
Glyph.hpp

```cpp
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied warranty.
6 // In no event will the authors be held liable for any damages arising from the use of this software.
7 //
8 // Permission is granted to anyone to use this software for any purpose,
9 // including commercial applications, and to alter it and redistribute it freely,
10 // subject to the following restrictions:
11 //
12 // 1. The origin of this software must not be misrepresented;
13 //    you must not claim that you wrote the original software.
14 // 2. Altered source versions must be plainly marked as such,
15 //    and must not be misrepresented as being the original software.
16 // 3. This notice may not be removed or altered from any source distribution.
17 //
18 #ifndef SFML_GLYPH_HPP
19 #define SFML_GLYPH_HPP
20 #endif
21
22 // Headers
23 #include <SFML/Graphics/Export.hpp>
24 #include <SFML/Graphics/Rect.hpp>
25
26 namespace sf
27 {
28
```
36 {
41 class SFML_GRAPHICS_API Glyph
42 {
43 public:
44
49     Glyph() : advance(0) {}
50
52     // Member data
54     float advance;
55     FloatRect bounds;
56     IntRect textureRect;
57    };
58
59 } // namespace sf
60
62 #endif // SFML_GLYPH_HPP
63
64
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source code.

#ifndef SFML_GRAPHICS_HPP
#define SFML_GRAPHICS_HPP

// Headers

#include <SFML/Window.hpp>
#include <SFML/Graphics/BlendMode.hpp>
#include <SFML/Graphics/CircleShape.hpp>
#include <SFML/Graphics/Color.hpp>

#endif
#include <SFML/Graphics/ConvexShape.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Font.hpp>
#include <SFML/Graphics/Glyph.hpp>
#include <SFML/Graphics/Image.hpp>
#include <SFML/Graphics/PrimitiveType.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/RenderTexture.hpp>
#include <SFML/Graphics/Renderer.hpp>
#include <SFML/Graphics/Shader.hpp>
#include <SFML/Graphics/Shape.hpp>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/Text.hpp>
#include <SFML/Graphics/Transform.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include <SFML/Graphics/Vertex.hpp>
#include <SFML/Graphics/VertexArray.hpp>
#include <SFML/Graphics/View.hpp>

#endif // SFML_GRAPHICS_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source.

#ifndef SFML_HTTP_HPP
#define SFML_HTTP_HPP

// Headers
#include <SFML/Network/Export.hpp>
#include <SFML/Network/IpAddress.hpp>
#include <SFML/Network/TcpSocket.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <SFML/System/Time.hpp>
#endif //SFML_HTTP_HPP
```cpp
#include <map>
#include <string>

namespace sf {

class SFML_NETWORK_API Http : NonCopyable {
public:

    class SFML_NETWORK_API Request {
public:

        enum Method {
            Get,
            Post,
            Head,
            Put,
            Delete
        };

        Request(const std::string& uri = "/", Method method = Get);

        void setField(const std::string& field, const std::string& value);

        void setMethod(Method method);

        void setUri(const std::string& uri);

        void setHttpVersion(unsigned int major, unsigned int minor);

        void setBody(const std::string& body);

    private:

        friend class Http;

        std::string prepare() const;

        bool hasField(const std::string& field) const;

    // Types

    typedef std::map<std::string, std::string> FieldTable;

    // Member data

    FieldTable m_fields;

    Method m_method;

    std::string m_uri;

    unsigned int m_majorVersion;

};

};

};
```
```cpp
unsigned int m_minorVersion;
std::string m_body;
}

class SFML_NETWORK_API Response
{
public:

    enum Status
    {
        // 2xx: success
        Ok = 200,
        Created = 201,
        Accepted = 202,
        NoContent = 204,
        ResetContent = 205,
        PartialContent = 206,

        // 3xx: redirection
        MultipleChoices = 300,
        MovedPermanently = 301,
        MovedTemporarily = 302,
        NotModified = 304,

        // 4xx: client error
        BadRequest = 400,
        Unauthorized = 401,
        Forbidden = 403,
        NotFound = 404,
        RangeNotSatisfiable = 407,

        // 5xx: server error
        InternalServerError = 500,
        NotImplemented = 501,
        BadGateway = 502,
        ServiceUnavailable = 503,
        GatewayTimeout = 504,
        VersionNotSupported = 505,

        // 10xx: SFML custom codes
        InvalidResponse = 1000,
        ConnectionFailed = 1001
    };

    Response();

    const std::string& getField(const std::string& field) const;

    Status getStatus() const;

    unsigned int getMajorHttpVersion() const;
```
unsigned int getMinorHttpVersion() const;

const std::string& getBody() const;

private:

friend class Http;

void parse(const std::string& data);

void parseFields(std::istream &in);

// Types
typedef std::map<std::string, std::string> FieldTable;

// Member data
FieldTable m_fields;
Status m_status;
unsigned int m_majorVersion;
unsigned int m_minorVersion;
std::string m_body;

};

Http();

Http(const std::string& host, unsigned short port = 0);

void setHost(const std::string& host, unsigned short port =

Response sendRequest(const Request& request, Time timeout =

private:

// Member data
TcpSocket m_connection;
IpAddress m_host;
std::string m_hostName;
unsigned short m_port;

} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_IMAGE_HPP
#define SFML_IMAGE_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <string>
#include <vector>
#endif // SFML_IMAGE_HPP
namespace sf
{

    class InputStream;

    class SFML_GRAPHICS_API Image
    {
        public:

            Image();
            ~Image();

            void create(unsigned int width, unsigned int height, const Color& color);
            void create(unsigned int width, unsigned int height, const Uint8* pixels);

            bool loadFromFile(const std::string& filename);
            bool loadFromMemory(const void* data, std::size_t size);
            bool loadFromStream(InputStream& stream);

            bool saveToFile(const std::string& filename) const;

            Vector2u getSize() const;

            void createMaskFromColor(const Color& color, Uint8 alpha = 0);

            void copy(const Image& source, unsigned int destX, unsigned int sourceRect = IntRect(0, 0, 0, 0), bool applyAlpha = false);

            void setPixel(unsigned int x, unsigned int y, const Color& color);

            Color getPixel(unsigned int x, unsigned int y) const;

            const Uint8* getPixelsPtr() const;

            void flipHorizontally();

            void flipVertically();

        private:

            // Member data

            Vector2u m_size;
            std::vector<Uint8> m_pixels;

            #ifdef SFML_SYSTEM_ANDROID
            void* m_stream;
            #endif

    };
} // namespace sf

#endif // SFML_IMAGE_HPP

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_INPUTSOUNDFILE_HPP
#define SFML_INPUTSOUNDFILE_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <SFML/System/Time.hpp>
#include <string>

#endif // SFML_INPUTSOUNDFILE_HPP
namespace sf
{
    class InputStream;
    class SoundFileReader;

    class SFML_AUDIO_API InputSoundFile : NonCopyable
    {
        public:

        InputSoundFile();
        ~InputSoundFile();

        bool openFromFile(const std::string& filename);
        bool openFromMemory(const void* data, std::size_t sizeInBytes);
        bool openFromStream(InputStream& stream);
        bool openForWriting(const std::string& filename, unsigned int sampleRate);

        Uint64 getSampleCount() const;
        unsigned int getChannelCount() const;
        unsigned int getSampleRate() const;
        Time getDuration() const;

        void seek(Uint64 sampleOffset);
        void seek(Time timeOffset);

        Uint64 read(Int16* samples, Uint64 maxCount);

        private:

        void close();

        // Member data
        SoundFileReader* m_reader;
        InputStream* m_stream;
        bool m_streamOwned;
        Uint64 m_sampleCount;
        unsigned int m_channelCount;
        unsigned int m_sampleRate;
    };
} // namespace sf
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
 InputStream.hpp

1  //
2  // SFML - Simple and Fast Multimedia Library
3  // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4  //
5  // This software is provided 'as-is', without any express or implied
6  // warranty.  In no event will the authors be held liable for any damages
7  // arising from the use of this software.
8  //
9  // Permission is granted to anyone to use this software for any
10  // purpose, including commercial applications, and to alter it and redist-
11  // tribute it freely, subject to the following restrictions:
12  //
13  // 1. The origin of this software must not be misrepresented;
14  //     you must not claim that you wrote the original software.
15  //     If you use this software in a product, an acknowledgment
16  //     in the product documentation would be appreciated but is not
17  //
18  // 2. Altered source versions must be plainly marked as such,
19  //     and must not be misrepresented as being the original software.
20  //
21  // 3. This notice may not be removed or altered from any source
22  //
23  //
24  #ifndef SFML_INPUTSTREAM_HPP
25  #define SFML_INPUTSTREAM_HPP
26  #endif  // SFML_INPUTSTREAM_HPP
27
28  // Headers
29  #include <SFML/Config.hpp>
30  #include <SFML/System/Export.hpp>
31
32  namespace sf
36  {
37     class SFML_SYSTEM_API InputStream
38     {
39         public:
40
41             virtual ~InputStream() {}
42
43             virtual Int64 read(void* data, Int64 size) = 0;
44
45             virtual Int64 seek(Int64 position) = 0;
46
47             virtual Int64 tell() = 0;
48
49             virtual Int64 getSize() = 0;
50        
51     };  // namespace sf
52
53  }  // SFML_INPUTSTREAM_HPP
54
55  #endif  // SFML_INPUTSTREAM_HPP
56
57
58  // SFML is licensed under the terms and conditions of the zlib/png license
59  Copyright © Laurent Gomila  ::  Documentation generated by doxygen
IpAddress.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty. In no event will the authors be held liable for any damages arising
7 // from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any purpose,
10 // including commercial applications, and to alter it and redistribute it
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 //    If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //    required.
18 //
19 // 2. Altered source versions must be plainly marked as such, and
20 //    must not be misrepresented as being the original software.
21 //
22 // 3. This notice may not be removed or altered from any source
23 //
24 #ifndef SFML_IPADDRESS_HPP
25 #define SFML_IPADDRESS_HPP
26 // Headers
27 #include <SFML/Network/Export.hpp>
28 #include <SFML/System/Time.hpp>
29 #include <istream>
30 #include <ostream>
31 #include <string>
namespace sf
{
    class SFML_NETWORK_API IpAddress
    {
    public:
        IpAddress();
        IpAddress(const std::string& address);
        IpAddress(const char* address);
        IpAddress(Uint8 byte0, Uint8 byte1, Uint8 byte2, Uint8 byte3);
        explicit IpAddress(Uint32 address);
        std::string toString() const;
        Uint32 toInteger() const;
        static IpAddress getLocalAddress();
        static IpAddress getPublicAddress(Time timeout = Time::Zero);
    
        // Static member data
        static const IpAddress None;
        static const IpAddress Any;
        static const IpAddress LocalHost;
        static const IpAddress Broadcast;
    
    private:
        friend SFML_NETWORK_API bool operator<(const IpAddress& left, const IpAddress& right);
        void resolve(const std::string& address);
    
        // Member data
        Uint32 m_address;
        bool m_valid;
    }
}

SFML_NETWORK_API bool operator == (const IpAddress& left, const IpAddress& right);
SFML_NETWORK_API bool operator != (const IpAddress& left, const IpAddress& right);
SFML_NETWORK_API bool operator < (const IpAddress& left, const IpAddress& right);
SFML_NETWORK_API bool operator > (const IpAddress& left, const IpAddress& right);
SFML_NETWORK_API bool operator <= (const IpAddress& left, const IpAddress& right);
SFML_NETWORK_API bool operator >=(const IpAddress& left, const IpAddress& right)
SFML_NETWORK_API std::istream& operator >> (std::istream& stream, IpAddress& value)
SFML_NETWORK_API std::ostream& operator << (std::ostream& stream, const IpAddress& value)

} // namespace sf

#endif // SFML_IPADDRESS_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Jeystick.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications,
// and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation
//    would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_JOYSTICK_HPP
#define SFML_JOYSTICK_HPP

// Headers
#include <SFML/Window/Export.hpp>
#include <SFML/System/String.hpp>

namespace sf {

```
class SFML_WINDOW_API Joystick {
public:

enum {
    Count = 8,
    ButtonCount = 32,
    AxisCount = 8
};

enum Axis {
    X,
    Y,
    Z,
    R,
    U,
    V,
    PovX,
    PovY
};

struct SFML_WINDOW_API Identification {
    Identification();
    String name;
    unsigned int vendorId;
    unsigned int productId;
};

static bool isConnected(unsigned int joystick);  
static unsigned int getButtonCount(unsigned int joystick); 
static bool hasAxis(unsigned int joystick, Axis axis);  
static bool isButtonPressed(unsigned int joystick, unsigned
static float getAxisPosition(unsigned int joystick, Axis axis);  
static Identification getIdentification(unsigned int joystick); 
static void update();
};
} // namespace sf

#endif // SFML_JOYSTICK_HPP
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_KEYBOARD_HPP
#define SFML_KEYBOARD_HPP

// Headers
#include <SFML/Window/Export.hpp>

namespace sf {


class SFML_WINDOW_API Keyboard
{
public:
    enum Key
    {
        Unknown = -1,
        A = 0,
        B,
        C,
        D,
        E,
        F,
        G,
        H,
        I,
        J,
        K,
        L,
        M,
        N,
        O,
        P,
        Q,
        R,
        S,
        T,
        U,
        V,
        W,
        X,
        Y,
        Z,
        Num0,
        Num1,
        Num2,
        Num3,
        Num4,
        Num5,
        Num6,
        Num7,
        Num8,
        Num9,
        Escape,
        LControl,
        LShift,
        LAlt,
        LSystem,
        RControl,
        RShift,
        RAAlt,
        RSystem,
F13, F14, F15, Pause, KeyCount;

static bool isKeyPressed(Key key);

static void setVirtualKeyboardVisible(bool visible);

// namespace sf

#endif // SFML_KEYBOARD_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Listener.hpp

```cpp
#include <SFML/Audio/Export.hpp>
#include <SFML/System/Vector3.hpp>

namespace sf
```
class SFML_AUDIO_API Listener
{
public:

    static void setGlobalVolume(float volume);
    static float getGlobalVolume();
    static void setPosition(float x, float y, float z);
    static void setPosition(const Vector3f& position);
    static Vector3f getPosition();
    static void setDirection(float x, float y, float z);
    static void setDirection(const Vector3f& direction);
    static Vector3f getDirection();
    static void setUpVector(float x, float y, float z);
    static void setUpVector(const Vector3f& upVector);
    static Vector3f getUpVector();
};

} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_LOCK_HPP
#define SFML_LOCK_HPP

// Headers
#include <SFML/System/Export.hpp>
#include <SFML/System/NonCopyable.hpp>

namespace sf {
class Mutex;

class SFML_SYSTEM_API Lock : NonCopyable
{
public:
    explicit Lock(Mutex& mutex);
    ~Lock();

private:
    Mutex& m_mutex;
};

} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied
// warranty.  In no event will the authors be held liable for any damages
// arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it
// freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment
//    in the product documentation would be appreciated but is not
//    required.
//
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source
//
#if defined(SFML_MAIN_HPP
#define SFML_MAIN_HPP
#endif

// Headers
#include <SFML/Config.hpp>

#if defined(SFML_SYSTEM_IOS)
// On iOS, we have no choice but to have our own main, so we need to rename the user one and call it later
#define main sfmlMain

#endif

#endif // SFML_MAIN_HPP

SFML is licensed under the terms and conditions of the zlib/png license

Copyright © Laurent Gomila :: Documentation generated by doxygen
mainpage.hpp

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
MemoryInputStream.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied warranty.
6 // In no event will the authors be held liable for any damages arising from the use of this software.
7 //
8 // Permission is granted to anyone to use this software for any purpose,
9 // including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
10 //
11 // 1. The origin of this software must not be misrepresented;
12 //      you must not claim that you wrote the original software.
13 //      If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
14 //
15 // 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
16 //
17 // 3. This notice may not be removed or altered from any source distribution.
18 //
19
20 #ifndef SFML_MEMORYINPUTSTREAM_HPP
21 #define SFML_MEMORYINPUTSTREAM_HPP
22
23 // Headers
24 #include <SFML/Config.hpp>
25 #include <SFML/System/InputStream.hpp>
26 #include <SFML/System/Export.hpp>
27 #include <cstdlib>
28
namespace sf {

class SFML_SYSTEM_API MemoryInputStream : public InputStream {

public:

    MemoryInputStream();

    void open(const void* data, std::size_t sizeInBytes);

    virtual Int64 read(void* data, Int64 size);

    virtual Int64 seek(Int64 position);

    virtual Int64 tell();

    virtual Int64 getSize();

private:

    // Member data
    const char* m_data;
    Int64 m_size;
    Int64 m_offset;
};

} // namespace sf

#endif // SFML_MEMORYINPUTSTREAM_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Mouse.hpp

```cpp
#ifndef SFML_MOUSE_HPP
#define SFML_MOUSE_HPP

namespace sf {


```
class Window;

class SFML_WINDOW_API Mouse
{
    public:
        enum Button
        {
            Left,
            Right,
            Middle,
            XButton1,
            XButton2,
        } ButtonCount
    
        enum Wheel
        {
            VerticalWheel,
            HorizontalWheel
        }

        static bool isButtonPressed(Button button);
        static Vector2i getPosition();
        static Vector2i getPosition(const Window& relativeTo);
        static void setPosition(const Vector2i& position);
        static void setPosition(const Vector2i& position, const Window&);
}

// namespace sf

#endif // SFML_MOUSE_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Music.hpp

1  //
2  // SFML - Simple and Fast Multimedia Library
3  // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4  //
5  // This software is provided 'as-is', without any express or implied
6  // In no event will the authors be held liable for any damages arising from
7  // This software is provided 'as-is', without any express or implied
8  // Permission is granted to anyone to use this software for any
9  // Permission is granted to anyone to use this software for any
10 // including commercial applications, and to alter it and redistribute it
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 // you must not claim that you wrote the original software.
15 // If you use this software in a product, an acknowledgment
16 // in the product documentation would be appreciated but is not
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 // and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 //
23
24 #ifndef SFML_MUSIC_HPP
25 #define SFML_MUSIC_HPP
26
27 // Headers
28 #include <SFML/Audio/Export.hpp>
29 #include <SFML/Audio/SoundStream.hpp>
30 #include <SFML/Audio/InputSoundFile.hpp>
31 #include <SFML/System/Mutex.hpp>
32 #include <SFML/System/Time.hpp>
#include <string>
#include <vector>

namespace sf {

  class InputStream;

  class SFML_AUDIO_API Music : public SoundStream {

      public:
        Music();
        ~Music();
        bool openFromFile(const std::string& filename);
        bool openFromMemory(const void* data, std::size_t sizeInBytes);
        bool openFromStream(InputStream& stream);
        Time getDuration() const;

      protected:
        virtual bool onGetData(Chunk& data);
        virtual void onSeek(Time timeOffset);

      private:
        void initialize();

    // Member data
    InputSoundFile m_file;
    Time m_duration;
    std::vector<Int16> m_samples;
    Mutex m_mutex;
  };

} // namespace sf

#endif // SFML_MUSIC_HPP

SFML is licensed under the terms and conditions of the zlib/png license.
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment
//    in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_MUTEX_HPP
#define SFML_MUTEX_HPP

namespace sf{
```cpp
{  
    namespace priv  
    {  
        class MutexImpl;  
    }  

    class SFML_SYSTEM_API Mutex : NonCopyable  
    {  
        public:  
        Mutex();  
        ~Mutex();  
        void lock();  
        void unlock();  
    
        private:  
        priv::MutexImpl* m_mutexImpl;  
    }  
    } // namespace sf

#endif // SFML_MUTEX_HPP
```
NativeActivity.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_NATIVEACTIVITY_HPP
#define SFML_NATIVEACTIVITY_HPP

// Headers
#include <SFML/System/Export.hpp>

#if !defined(SFML_SYSTEM_ANDROID)

#endif
```
#error NativeActivity.hpp: This header is Android only.
#endif

struct ANativeActivity;

namespace sf {
SFML_SYSTEM_API ANativeActivity* getNativeActivity();
} // namespace sf

} // namespace sf

#endif // SFML_NATIVEACTIVITY_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
# SFML 2.4.2

Network.hpp

```cpp
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty.  In no event will the authors be held liable for any damages
7 // arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redistri
11 // all restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 //    If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 //    and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 //
23 #ifndef SFML_NETWORK_HPP
24 #define SFML_NETWORK_HPP
25 // Headers
26 #include <SFML/System.hpp>
27 #include <SFML/Network/Ftp.hpp>
28 #include <SFML/Network/Http.hpp>
29 #include <SFML/Network/IpAddress.hpp>
```
#include <SFML/Network/ Packet.hpp>
#include <SFML/Network/ Socket.hpp>
#include <SFML/Network/ SocketHandle.hpp>
#include <SFML/Network/ SocketSelector.hpp>
#include <SFML/Network/ TcpListener.hpp>
#include <SFML/Network/ TcpSocket.hpp>
#include <SFML/Network/ UdpSocket.hpp>

#endif // SFML_NETWORK_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied
// warranty.  In no event will the authors be held liable for any damages
// arising from the use of this software.
//
// Permission is granted to anyone to use this software for any
// purpose, including commercial applications, and to alter it and
// redistribute it freely, subject to the following restrictions:
//
//  1. The origin of this software must not be misrepresented;
//     you must not claim that you wrote the original software.
//     If you use this software in a product, an acknowledgment
//     in the product documentation would be appreciated but is not
//     required.
//  2. Altered source versions must be plainly marked as such,
//     and must not be misrepresented as being the original software.
//  3. This notice may not be removed or altered from any source.

#ifndef SFML_NONCOPYABLE_HPP
#define SFML_NONCOPYABLE_HPP

namespace sf {

// Headers
#include <SFML/System/Export.hpp>

namespace sf {

class SFML_SYSTEM_API NonCopyable
{
protected:
    NonCopyable() {}

private:
    NonCopyable(const NonCopyable&);
    NonCopyable& operator =(const NonCopyable&);
};
// namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_OPENGL_HPP
#define SFML_OPENGL_HPP

#include <SFML/Config.hpp>

#if defined(SFML_SYSTEM_WINDOWS)

#include <SFML/Config.hpp>
#endif

#ifdef(SFML_SYSTEM_WINDOWS)

#endif
#endif
// The Visual C++ version of gl.h uses WINGDIAPI and APIENTRY but
doesn't define them

#ifdef _MSC_VER
    #include <windows.h>
#endif

#include <GL/gl.h>

#elif defined(SFML_SYSTEM_LINUX) || defined(SFML_SYSTEM_FREEBSD)
    #if defined(SFML_OPENGL_ES)
        #include <GLES/gl.h>
        #include <GLES/glext.h>
    #else
        #include <GL/gl.h>
    #endif

#elif defined(SFML_SYSTEM_MACOS)
    #include <OpenGL/gl.h>

#elif defined(SFML_SYSTEM_IOS)
    #include <OpenGLES/ES1/gl.h>
    #include <OpenGLES/ES1/glext.h>

#elif defined(SFML_SYSTEM_ANDROID)
    #include <GLES/gl.h>
    #include <GLES/glext.h>

    // We're not using OpenGL ES 2+ yet, but we can use the sRGB
    #include <GLES2/gl2ext.h>

#endif // SFML_OPENGL_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_OUTPUTSOUNDFILE_HPP
#define SFML_OUTPUTSOUNDFILE_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <string>

#endif // SFML_OUTPUTSOUNDFILE_HPP
namespace sf
{
class SoundFileWriter;

class SFML_AUDIO_API OutputSoundFile : NonCopyable
{
public:

OutputSoundFile();
~OutputSoundFile();

bool openFromFile(const std::string& filename, unsigned int channelCount);

void write(const Int16* samples, Uint64 count);

private:

void close();

// Member data
SoundFileWriter* m_writer;

};

} // namespace sf

#endif // SFML_OUTPUTSOUNDFILE_HPP
Packet.hpp

```c++
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_PACKET_HPP
#define SFML_PACKET_HPP

// Headers
#include <SFML/Network/Export.hpp>
#include <string>
#include <vector>
```

```
namespace sf
{
class String;
class TcpSocket;
class UdpSocket;

class SFML_NETWORK_API Packet
{
    // A bool-like type that cannot be converted to integer or pointer types
    typedef bool (Packet::*BoolType)(std::size_t);

public:

    Packet();
    virtual ~Packet();
    void append(const void* data, std::size_t sizeInBytes);
    void clear();

    const void* getData() const;
    std::size_t getDataSize() const;
    bool endOfPacket() const;

public:

    operator BoolType() const;

    Packet& operator >> (bool& data);
    Packet& operator >> (Int8& data);
    Packet& operator >> (Uint8& data);
    Packet& operator >> (Int16& data);
    Packet& operator >> (Uint16& data);
    Packet& operator >> (Int32& data);
    Packet& operator >> (Uint32& data);
    Packet& operator >> (Int64& data);
    Packet& operator >> (Uint64& data);
    Packet& operator >> (float& data);
    Packet& operator >> (double& data);
    Packet& operator >> (char* data);
    Packet& operator >> (std::string& data);
    Packet& operator >> (wchar_t* data);
    Packet& operator >> (std::wstring& data);
    Packet& operator >> (String& data);

    Packet& operator << (bool data);
    Packet& operator << (Int8 data);
    Packet& operator << (Uint8 data);
    Packet& operator << (Int16 data);
```
202     Packet& operator <<(Uint16     data);
203     Packet& operator <<(Int32      data);
204     Packet& operator <<(Uint32     data);
205     Packet& operator <<(Int64      data);
206     Packet& operator <<(Uint64     data);
207     Packet& operator <<(float      data);
208     Packet& operator <<(double     data);
209     Packet& operator <<(const char* data);
210     Packet& operator <<(const std::string& data);
211     Packet& operator <<(const wchar_t* data);
212     Packet& operator <<(const std::wstring& data);
213     Packet& operator <<(const String& data);
214
protected:
215     friend class TcpSocket;
216     friend class UdpSocket;
217     virtual const void* onSend(std::size_t& size);
218     virtual void onReceive(const void* data, std::size_t size);
219
private:
220     bool operator ==((const Packet& right) const;
221     bool operator !=((const Packet& right) const;
222     bool checkSize(std::size_t size);
223     // Member data
224     std::vector<char> m_data;
225     std::size_t m_readPos;
226     std::size_t m_sendPos;
227     bool m_isValid;
228
229 } // namespace sf
230
231 #endif // SFML_PACKET_HPP
```
/* SFML - Simple and Fast Multimedia Library */
/* Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org) */
/* This software is provided 'as-is', without any express or implied */
/* warranty. In no event will the authors be held liable for any damages */
/* arising from the use of this software. */
/* Permission is granted to anyone to use this software for any purpose, */
/* including commercial applications, and to alter it and redistribute */
/* it freely, subject to the following restrictions: */
/* 1. The origin of this software must not be misrepresented; */
/*  you must not claim that you wrote the original software. */
/* 2. Altered source versions must be plainly marked as such, */
/*  and must not be misrepresented as being the original software. */
/* 3. This notice may not be removed or altered from any source */
#include <SFML/PRIMITIVETYPE.hpp>
namespace sf {
enum PrimitiveType {
  Points,
  Lines,
};
LineStrip,
Triangles,
TriangleStrip,
TriangleFan,
Quads,

// Deprecated names
LinesStrip   = LineStrip,
TrianglesStrip = TriangleStrip,
TrianglesFan   = TriangleFan

};
// namespace sf

#endif // SFML_PRIMITIVETYPE_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Rect.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_RECT_HPP
#define SFML_RECT_HPP

// Headers
#include <SFML/System/Vector2.hpp>
#include <algorithm>

namespace sf
```
template <typename T>
class Rect {
    public:
        Rect();
        Rect(T rectLeft, T rectTop, T rectWidth, T rectHeight);
        Rect(const Vector2<T>& position, const Vector2<T>& size);

        template <typename U>
        explicit Rect(const Rect<U>& rectangle);

        bool contains(T x, T y) const;
        bool contains(const Vector2<T>& point) const;

        bool intersects(const Rect<T>& rectangle) const;
        bool intersects(const Rect<T>& rectangle, Rect<T>& intersection);

    // Member data
    T left;
    T top;
    T width;
    T height;
};

template <typename T>
bool operator == (const Rect<T>& left, const Rect<T>& right);

template <typename T>
bool operator != (const Rect<T>& left, const Rect<T>& right);

#include <SFML/Graphics/Rect.inl>

// Create typedefs for the most common types
typedef Rect<int> IntRect;
typedef Rect<float> FloatRect;

} // namespace sf

#endif // SFML_RECT_HPP
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
RectangeShape.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_RECTANGLESHAPE_HPP
#define SFML_RECTANGLESHAPE_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Shape.hpp>

namespace sf {
```
```
class SFML_GRAPHICS_API RectangleShape : public Shape
{
public:

    explicit RectangleShape(const Vector2f& size = Vector2f(0, 0));

    void setSize(const Vector2f& size);

    const Vector2f& getSize() const;

    virtual std::size_t getPointCount() const;

    virtual Vector2f getPoint(std::size_t index) const;

private:

    // Member data

    Vector2f m_size;

} // namespace sf

#endif // SFML_RECTANGLESHAPE_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment
//    in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source.

#ifndef SFML_RENDERSTATES_HPP
#define SFML_RENDERSTATES_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/BlendMode.hpp>
#include <SFML/Graphics/Transform.hpp>

#endif // SFML_RENDERSTATES_HPP
namespace sf
{
class Shader;
class Texture;

class SFML_GRAPHICS_API RenderStates
{
public:
    RenderStates();
    RenderStates(const BlendMode& theBlendMode);
    RenderStates(const Transform& theTransform);
    RenderStates(const Texture* theTexture);
    RenderStates(const Shader* theShader);
    RenderStates(const BlendMode& theBlendMode, const Transform& theTransform, const Texture* theTexture, const Shader* theShader);

    // Static member data
    static const RenderStates Default;

    // Member data
    BlendMode blendMode;
    Transform transform;
    const Texture* texture;
    const Shader* shader;
};

} // namespace sf

#endif // SFML_RENDERSTATES_HPP
SFML 2.4.2

RenderTarget.hpp

```cpp
1 //
3 // SFML - Simple and Fast Multimedia Library
4 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
5 //
6 // This software is provided 'as-is', without any express or implied
7 // warranty.  In no event will the authors be held liable for any damages arising
8 // from the use of this software.
9 //
10 // Permission is granted to anyone to use this software for any purpose,
11 // including commercial applications, and to alter it and redistribute it
12 // subject to the following restrictions:
13 //
14 // 1. The origin of this software must not be misrepresented;
15 //    you must not claim that you wrote the original software.
16 //    If you use this software in a product, an acknowledgment
17 //    in the product documentation would be appreciated but is not
18 //    required.
19 //
20 // 2. Altered source versions must be plainly marked as such,
21 //    and must not be misrepresented as being the original software.
22 //
23 // 3. This notice may not be removed or altered from any source
24 //
25 #ifndef SFML_RENDERTARGET_HPP
26 #define SFML_RENDERTARGET_HPP
27 // Headers
28 #include <SFML/Graphics/Export.hpp>
29 #include <SFML/Graphics/Color.hpp>
30 #include <SFML/Graphics/Rect.hpp>
31 #include <SFML/Graphics/View.hpp>
32 #include <SFML/Graphics/Transform.hpp>
```
#include <SFML/Graphics/BlendMode.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/PrimitiveType.hpp>
#include <SFML/Graphics/Vertex.hpp>
#include <SFML/System/NonCopyable.hpp>

namespace sf
{
    class Drawable;
}

SFML_GRAPHICS_API RenderTarget : NonCopyable
{
    public:

    virtual ~RenderTarget();

    void clear(const Color& color = Color(0, 0, 0, 255));

    void setView(const View& view);

    const View& getView() const;

    const View& getDefaultView() const;

    IntRect getViewport(const View& view) const;

    Vector2f mapPixelToCoords(const Vector2i& point) const;

    Vector2f mapPixelToCoords(const Vector2i& point, const View& view) const;

    Vector2i mapCoordsToPixel(const Vector2f& point) const;

    Vector2i mapCoordsToPixel(const Vector2f& point, const View& view) const;

    void draw(const Drawable& drawable, const RenderStates& states);

    void draw(const Vertex* vertices, std::size_t vertexCount, PrimitiveType type, const RenderStates& states = RenderStates::Default);

    virtual Vector2u getSize() const = 0;

    void pushGLStates();

    void popGLStates();

    void resetGLStates();

    protected:

   RenderTarget();
void initialize();

private:

void applyCurrentView();
void applyBlendMode(const BlendMode& mode);
void applyTransform(const Transform& transform);
void applyTexture(const Texture* texture);
void applyShader(const Shader* shader);

virtual bool activate(bool active) = 0;

struct StatesCache
{
    enum {VertexCacheSize = 4};

    bool glStatesSet;
    bool viewChanged;
    BlendMode lastBlendMode;
    Uint64 lastTextureId;
    bool useVertexCache;
    Vertex vertexCache[VertexCacheSize];
};

// Member data
View m_defaultView;
View m_view;
StatesCache m_cache;

} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen

```
/**SFML 2.4.2**

**RenderTexture.hpp**

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_RENDERTEXTURE_HPP
#define SFML_RENDERTEXTURE_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/Graphics/RenderTarget.hpp>

#endif // SFML_RENDERTEXTURE_HPP
```
namespace sf
{
namespace priv
{
    class RenderTextureImpl;
}

class SFML_GRAPHICS_API RenderTexture : public RenderTarget
{
public:
    RenderTexture();
    virtual ~RenderTexture();

    bool create(unsigned int width, unsigned int height, bool depthBuffer = false);
    void setSmooth(bool smooth);
    bool isSmooth() const;
    void setRepeated(bool repeated);
    bool isRepeated() const;
    bool generateMipmap();
    bool setActive(bool active = true);
    void display();
    virtual Vector2u getSize() const;
    const Texture& getTexture() const;

private:
    virtual bool activate(bool active);
    // Member data
    priv::RenderTextureImpl* m_impl;
    Texture m_texture;
};

// namespace sf
}
#endif // SFML_RENDERTEXTURE_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied
// warranty.  In no event will the authors be held liable for any damages arising
// from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it
// freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source

#define SFML_RENDERWINDOW_HPP

#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/Image.hpp>
#include <SFML/Window/Window.hpp>
#include <string>
namespace sf
{
    class SFML_GRAPHICS_API RenderWindow : public Window, public RenderTarget
    {
        public:
           RenderWindow();
           RenderWindow(VideoMode mode, const String& title, Uint32 style, ContextSettings& settings = ContextSettings());
            explicit RenderWindow(WindowHandle handle, const ContextSettings& settings = ContextSettings());
            virtual ~RenderWindow();
            virtual Vector2u getSize() const;
            SFML_DEPRECATED Image capture() const;
            protected:
                virtual void onCreate();
                virtual void onResize();
                private:
                    virtual bool activate(bool active);
    }; // namespace sf
}
#endif // SFML_RENDERWINDOW_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied
// warranty.  In no event will the authors be held liable for any damages
// arising from the use of this software.
//
// Permission is granted to anyone to use this software for any
// purpose, including commercial applications, and to alter it and redist
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment
// in the product documentation would be appreciated but is not
// required.
//
// 2. Altered source versions must be plainly marked as such,
// and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source

#ifndef SFML_SENSOR_HPP
#define SFML_SENSOR_HPP

// Headers
#include <SFML/Window/Export.hpp>
#include <SFML/System/Vector3.hpp>
#include <SFML/System/Time.hpp>

#endif // SFML_SENSOR_HPP
namespace sf
{

class SFML_WINDOW_API Sensor
{
    public:
        enum Type
        {
            Accelerometer,
            Gyroscope,
            Magnetometer,
            Gravity,
            UserAcceleration,
            Orientation,

            Count
        };

        static bool isAvailable(Type sensor);

        static void setEnabled(Type sensor, bool enabled);

        static Vector3f getValue(Type sensor);
    }

} // namespace sf

#endif // SFML_SENSOR_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Shader.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_SHADER_HPP
#define SFML_SHADER_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Glsl.hpp>
#include <SFML/Window/GlResource.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <SFML/System/Vector2.hpp>
```
```cpp
#include <SFML/System/Vector3.hpp>
#include <map>
#include <string>

namespace sf {
    class Color;
    class InputStream;
    class Texture;
    class Transform;

    class SFML_GRAPHICS_API Shader : GlResource, NonCopyable {
    public:
        enum Type {
            Vertex,
            Geometry,
            Fragment
        };

        struct CurrentTextureType {};

        static CurrentTextureType CurrentTexture;

    public:
        Shader();
        ~Shader();

        bool loadFromFile(const std::string& filename, Type type);
        bool loadFromFile(const std::string& vertexShaderFilename, const std::string& fragmentShaderFilename);
        bool loadFromMemory(const std::string& vertexShader, const std::string& fragmentShader);
        bool loadFromStream(InputStream& stream, Type type);
    }
```
bool loadFromStream(InputStream& vertexShaderStream, InputStream& fragmentShaderStream);

void setUniform(const std::string& name, float x);
void setUniform(const std::string& name, const Glsl::Vec2& vector);
void setUniform(const std::string& name, const Glsl::Vec3& vector);
void setUniform(const std::string& name, const Glsl::Vec4& vector);
void setUniform(const std::string& name, int x);
void setUniform(const std::string& name, const Glsl::Ivec2& vector);
void setUniform(const std::string& name, const Glsl::Ivec3& vector);
void setUniform(const std::string& name, const Glsl::Ivec4& vector);
void setUniform(const std::string& name, bool x);
void setUniform(const std::string& name, const Glsl::Bvec2& vector);
void setUniform(const std::string& name, const Glsl::Bvec3& vector);
void setUniform(const std::string& name, const Glsl::Bvec4& vector);
void setUniform(const std::string& name, const Glsl::Mat3& matrix);
void setUniform(const std::string& name, const Glsl::Mat4& matrix);
void setUniform(const std::string& name, CurrentTextureType);
void setUniformArray(const std::string& name, const float* scalarArray, std::size_t length);
void setUniformArray(const std::string& name, const Glsl::Vec2& length);
void setUniformArray(const std::string& name, const Glsl::Vec3& length);
void setUniformArray(const std::string& name, const Glsl::Vec4& length);
void setUniformArray(const std::string& name, const Glsl::Mat3& length);
void setUniformArray(const std::string& name, const Glsl::Mat4& length);
SFML_DEPRECATED
void setParameter(const std::string& name, float w);

SFML_DEPRECATED
void setParameter(const std::string& name, const unsigned int getNativeHandle() const;

static void bind(const Shader* shader);
static bool isAvailable();
static bool isGeometryAvailable();

private:
bool compile(const char* vertexShaderCode, const char* geometryShaderCode, const char* fragmentShaderCode);

void bindTextures() const;
int getUniformLocation(const std::string& name);

struct UniformBinder;

// Types
typedef std::map<int, const Texture*> TextureTable;
typedef std::map<std::string, int> UniformTable;

// Member data
unsigned int m_shaderProgram;
int m_currentTexture;
TextureTable m_textures;
UniformTable m_uniforms;
};
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
Shape.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty.  In no event will the authors be held liable for any damages
7 // arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redis-
11 // tribute it freely, subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 // 2. If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //    required.
18 // 3. This notice may not be removed or altered from any source
19 //
20 #ifndef SFML_SHAPE_HPP
21 #define SFML_SHAPE_HPP
22
23 #include <SFML/Graphics/Export.hpp>
24 #include <SFML/Graphics/Drawable.hpp>
25 #include <SFML/Graphics/Transformable.hpp>
26 #include <SFML/Graphics/VertexArray.hpp>
27 #include <SFML/System/Vector2.hpp>
namespace sf
{
    class SFML_GRAPHICS_API Shape : public Drawable, public Transformable
    {
        public:
            virtual ~Shape();

            void setTexture(const Texture* texture, bool resetRect = false);
            void setTextureRect(const IntRect& rect);
            void setFillColor(const Color& color);
            void setOutlineColor(const Color& color);
            void setOutlineThickness(float thickness);
            const Texture* getTexture() const;
            const IntRect& getTextureRect() const;
            const Color& getFillColor() const;
            const Color& getOutlineColor() const;
            float getOutlineThickness() const;
            virtual std::size_t getPointCount() const = 0;
            virtual Vector2f getPoint(std::size_t index) const = 0;
            FloatRect getLocalBounds() const;
            FloatRect getGlobalBounds() const;
        protected:
            Shape();
            void update();
        private:
            virtual void draw(RenderTarget& target, RenderStates states);
            void updateFillColors();
            void updateTexCoords();
    }
void updateOutline();
void updateOutlineColors();

private:
  // Member data
  const Texture* m_texture;
  IntRect m_textureRect;
  Color m_fillColor;
  Color m_outlineColor;
  float m_outlineThickness;
  VertexArray m_vertices;
  VertexArray m_outlineVertices;
  FloatRect m_insideBounds;
  FloatRect m_bounds;

} // namespace sf

#endif // SFML_SHAPE_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SLEEP_HPP
#define SFML_SLEEP_HPP

#include <SFML/System/Export.hpp>
#include <SFML/System/Time.hpp>

namespace sf
{ void SFML_SYSTEM_API sleep(Time duration);
} // namespace sf

#endif // SFML_SLEEP_HPP

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_SOCKET_HPP
#define SFML_SOCKET_HPP

// Headers
#include <SFML/Network/Export.hpp>
#include <SFML/Network/SocketHandle.hpp>
#include <SFML/System/NonCopyable.hpp>
#include<vector>

# ifndef SFML_SOCKET_HPP
# define SFML_SOCKET_HPP

// Headers
#include <SFML/Network/Export.hpp>
#include <SFML/Network/SocketHandle.hpp>
#include <SFML/System/NonCopyable.hpp>
#include<vector>
namespace sf
{
    class SocketSelector;
}

class SFML_NETWORK_API Socket : NonCopyable
{
    public:
        enum Status
        {
            Done,
            NotReady,
            Partial,
           Disconnected,
            Error
        };

        enum
        {
            AnyPort = 0
        };

    public:
        virtual ~Socket();
        void setBlocking(bool blocking);
        bool isBlocking() const;

    protected:
        enum Type
        {
            Tcp,
            Udp
        };

    Socket(Type type);
    SocketHandle getHandle() const;
    void create();
    void create(SocketHandle handle);
    void close();

    private:
        friend class SocketSelector;
// Member data
Type m_type;
SocketHandle m_socket;
bool m_isBlocking;
}

} // namespace sf

#endif // SFML_SOCKET_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
SocketHandle.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SOCKETHANDLE_HPP
#define SFML_SOCKETHANDLE_HPP

// Headers
#include <SFML/Config.hpp>

#if defined(SFML_SYSTEM_WINDOWS)
#include <basetsd.h>
#endif
#endif
```

namespace sf
{
    // Define the low-level socket handle type, specific to each platform
    #if defined(SFML_SYSTEM_WINDOWS)
        typedef UINT_PTR SocketHandle;
    #else
        typedef int SocketHandle;
    #endif

} // namespace sf

#endif // SFML_SOCKETHANDLE_HPP

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SOCKETSELECTOR_HPP
#define SFML_SOCKETSELECTOR_HPP

// Headers
#include <SFML/Network/Export.hpp>
#include <SFML/System/Time.hpp>

namespace sf{
class Socket;

class SFML_NETWORK_API SocketSelector {
public:
    SocketSelector();
    SocketSelector(const SocketSelector& copy);
    ~SocketSelector();
    void add(Socket& socket);
    void remove(Socket& socket);
    void clear();
    bool wait(Time timeout = Time::Zero);
    bool isReady(Socket& socket) const;
    SocketSelector& operator =(const SocketSelector& right);
private:
    struct SocketSelectorImpl;
    // Member data
    SocketSelectorImpl* m_impl;
};

// namespace sf
SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Sound.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SOUND_HPP
#define SFML_SOUND_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/Audio/SoundSource.hpp>
#include <SFML/System/Time.hpp>
#include <cstdlib>
```

namespace sf {

class SoundBuffer;

class SFML_AUDIO_API Sound : public SoundSource {
{
public:

Sound();

explicit Sound(const SoundBuffer& buffer);

Sound(const Sound& copy);

~Sound();

void play();

void pause();

void stop();

void setBuffer(const SoundBuffer& buffer);

void setLoop(bool loop);

void setPlayingOffset(Time timeOffset);

const SoundBuffer* getBuffer() const;

bool getLoop() const;

Time getPlayingOffset() const;

Status getStatus() const;

Sound& operator =(const Sound& right);

void resetBuffer();

private:

// Member data

const SoundBuffer* m_buffer;

};

} // namespace sf

#endif // SFML_SOUND_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SOUNDBUFFER_HPP
#define SFML_SOUNDBUFFER_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/Audio/AIResource.hpp>
#include <SFML/System/Time.hpp>
#include <string>
#include <vector>
#endif // SFML_SOUNDBUFFER_HPP
```cpp
#include <set>

namespace sf {

class Sound;

class InputSoundFile;

class InputStream;

class SFML_AUDIO_API SoundBuffer : AlResource {

public:

    SoundBuffer();

    SoundBuffer(const SoundBuffer& copy);

    ~SoundBuffer();

    bool loadFromFile(const std::string& filename);

    bool loadFromMemory(const void* data, std::size_t sizeInBytes);

    bool loadFromStream(InputStream& stream);

    bool loadFromSamples(const Int16* samples, Uint64 sampleCount,
                          unsigned int sampleRate);

    bool saveToFile(const std::string& filename) const;

    const Int16* getSamples() const;

    Uint64 getSampleCount() const;

    unsigned int getSampleRate() const;

    unsigned int getChannelCount() const;

    Time getDuration() const;

    SoundBuffer& operator =(const SoundBuffer& right);

private:

    friend class Sound;

    bool initialize(InputSoundFile& file);

    bool update(unsigned int channelCount, unsigned int sampleRate);

    void attachSound(Sound* sound) const;

};
```
void detachSound(Sound* sound) const;

// Types
typedef std::set<Sound*> SoundList;

// Member data
unsigned int m_buffer;
std::vector<Int16> m_samples;
Time m_duration;
mutable SoundList m_sounds;
};
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_SOUNDBUFFERRECORORDER_HPP
#define SFML_SOUNDBUFFERRECORORDER_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/Audio/SoundBuffer.hpp>
#include <SFML/Audio/SoundRecorder.hpp>
#include <vector>
namespace sf
{
class SFML_AUDIO_API SoundBufferRecorder : public SoundRecorder
{
public:

~SoundBufferRecorder();

const SoundBuffer& getBuffer() const;

protected:

virtual bool onStart();

virtual bool onProcessSamples(const Int16* samples, std::size_t sampleCount);

virtual void onStop();

private:

// Member data
std::vector<Int16> m_samples;

SoundBuffer m_buffer;

};

} // namespace sf

#endif // SFML_SOUNDBUFFERRECORDER_HPP
SoundFileFactory.hpp

1  //
2  // SFML - Simple and Fast Multimedia Library
3  // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4  //
5  // This software is provided 'as-is', without any express or implied
6  // warranty.  In no event will the authors be held liable for any damages
7  // arising from the use of this software.
8  //
9  // Permission is granted to anyone to use this software for any
10  // purpose, including commercial applications, and to alter it and redistri-
11  // bution, subject to the following restrictions:
12  //
13  // 1. The origin of this software must not be misrepresented;
14  //     you must not claim that you wrote the original software.
15  //     If you use this software in a product, an acknowledgment
16  //     in the product documentation would be appreciated but is not
17  //     required.
18  // 2. Altered source versions must be plainly marked as such,
19  //     and must not be misrepresented as being the original software.
20  //
21  // 3. This notice may not be removed or altered from any source
22  //
23  #ifndef SFML_SOUNDFILEFACTORY_HPP
24  #define SFML_SOUNDFILEFACTORY_HPP
25  
26  // Headers
27  #include <SFML/Audio/Export.hpp>
28  #include <string>
29  #include <vector>
namespace sf
{
class InputStream;
class SoundFileReader;
class SoundFileWriter;

class SFML_AUDIO_API SoundFileFactory
{
public:
    template <typename T>
    static void registerReader();

template <typename T>
    static void unregisterReader();

template <typename T>
    static void registerWriter();

template <typename T>
    static void unregisterWriter();

    static SoundFileReader* createReaderFromFilename(const std::string& filename);

    static SoundFileReader* createReaderFromMemory(const void* data, std::size_t sizeInBytes);

    static SoundFileReader* createReaderFromStream(InputStream& stream);

    static SoundFileWriter* createWriterFromFilename(const std::string& filename);

private:
    // Types
    struct ReaderFactory
    {
        bool (*check)(InputStream&);
        SoundFileReader* (*create)();
    };
    typedef std::vector<ReaderFactory> ReaderFactoryArray;

    struct WriterFactory
    {
        bool (*check)(const std::string&);
        SoundFileWriter* (*create)();
    };
    typedef std::vector<WriterFactory> WriterFactoryArray;

    // Static member data
    static ReaderFactoryArray s_readers;
    static WriterFactoryArray s_writers;
};
```cpp
} // namespace sf
#include <SFML/Audio/SoundFileFactory.inl>
#endif // SFML_SOUNDFILEFACTORY_HPP
```
SoundFileReader.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SOUNDFILEREADER_HPP
#define SFML_SOUNDFILEREADER_HPP

#include <SFML/Audio/Export.hpp>
#include <string>

namespace sf
```
class InputStream;

class SFML_AUDIO_API SoundFileReader
{
public:
    struct Info
    {
        Uint64 sampleCount;
        unsigned int channelCount;
        unsigned int sampleRate;
    };

    virtual ~SoundFileReader() {}

    virtual bool open(InputStream& stream, Info& info) = 0;

    virtual void seek(Uint64 sampleOffset) = 0;

    virtual Uint64 read(Int16* samples, Uint64 maxCount) = 0;
};

// namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications,
// and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_SOUNDFILEWRITER_HPP
#define SFML_SOUNDFILEWRITER_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <string>

namespace sf {
36 {  
41 class SFML_AUDIO_API SoundFileWriter  
42 {  
43 public:  
44  
49     virtual ~SoundFileWriter() {}  
50     virtual bool open(const std::string& filename, unsigned int channelCount) = 0;  
61     virtual void write(const Int16* samples, Uint64 count) = 0;  
70   };  
71 } // namespace sf  
72  
73 #endif // SFML_SOUNDFILEWRITER_HPP  
74  
75  
76
SoundRecorder.hpp

```cpp
#ifndef SFML_SOUNDRECORDER_HPP
#define SFML_SOUNDRECORDER_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/Audio/A1Resource.hpp>
#include <SFML/System/Thread.hpp>
#include <SFML/System/Time.hpp>
#include <vector>
```

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#define SFML_SOUNDRECORDER_HPP
```
36#include <string>

39namespace sf
40{
45class SFML_AUDIO_API SoundRecorder : AlResource
46{
47public:
48    virtual ~SoundRecorder();

52    bool start(unsigned int sampleRate = 44100);
53    void stop();

57    unsigned int getSampleRate() const;

62    static std::vector<std::string> getAvailableDevices();

66    static std::string getDefaultDevice();

71    bool setDevice(const std::string& name);

75    const std::string& getDevice() const;

79    void setChannelCount(unsigned int channelCount);

84    unsigned int getChannelCount() const;

89    static bool isAvailable();

94protected:
95    SoundRecorder();

100    void setProcessingInterval(Time interval);

105    virtual bool onStart();

110    virtual bool onProcessSamples(const Int16* samples, std::size_t sampleCount);

115    virtual void onStop();

120private:
121    void record();

126    void processCapturedSamples();

131    void cleanup();

136    // Member data
Thread m_thread;
std::vector<Int16> m_samples;
unsigned int m_sampleRate;
Time m_processingInterval;
bool m_isCapturing;
std::string m_deviceName;
unsigned int m_channelCount;
};
} // namespace sf
#endif // SFML_SOUNDRECORDER_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_SOUNDSOURCE_HPP
#define SFML_SOUNDSOURCE_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/Audio/ALResource.hpp>
#include <SFML/System/Vector3.hpp>

#endif // SFML_SOUNDSOURCE_HPP
namespace sf
{
    class SFML_AUDIO_API SoundSource : A1Resource
    {
        public:

        enum Status
        {
            Stopped,
            Paused,
            Playing
        };

        SoundSource(const SoundSource& copy);

        virtual ~SoundSource();

        void setPitch(float pitch);

        void setVolume(float volume);

        void setPosition(float x, float y, float z);

        void setPosition(const Vector3f& position);

        void setRelativeToListener(bool relative);

        void setMinDistance(float distance);

        void setAttenuation(float attenuation);

        float getPitch() const;

        float getVolume() const;

        Vector3f getPosition() const;

        bool isRelativeToListener() const;

        float getMinDistance() const;

        float getAttenuation() const;

        SoundSource& operator = (const SoundSource& right);

        protected:

        SoundSource();

        Status getStatus() const;

        // Member data
unsigned int m_source;
};
} // namespace sf

#endif // SFML_SOUNDSOURCE_HPP

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from
// the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it,
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment in the
// product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such, and
// must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SOUNDSTREAM_HPP
#define SFML_SOUNDSTREAM_HPP

// Headers
#include <SFML/Audio/Export.hpp>
#include <SFML/Audio/SoundSource.hpp>
#include <SFML/System/Thread.hpp>
#include <SFML/System/Time.hpp>
#include <SFML/System/Mutex.hpp>

#include <cstdlib>
namespace sf {
    class SFML_AUDIO_API SoundStream : public SoundSource {
    public:
        struct Chunk {
            const Int16* samples;
            std::size_t sampleCount;
        }
        virtual ~SoundStream();
        void play();
        void pause();
        void stop();
        unsigned int getChannelCount() const;
        unsigned int getSampleRate() const;
        Status getStatus() const;
        void setPlayingOffset(Time timeOffset);
        Time getPlayingOffset() const;
        void setLoop(bool loop);
        bool getLoop() const;
    protected:
        SoundStream();
        void initialize(unsigned int channelCount, unsigned int sampleRate);
        virtual bool onGetData(Chunk& data) = 0;
        virtual void onSeek(Time timeOffset) = 0;
    private:
        void streamData();
        bool fillAndPushBuffer(unsigned int bufferNum);
bool fillQueue();
void clearQueue();

enum {
    BufferCount = 3
};

// Member data
Thread m_thread;
mutable Mutex m_threadMutex;
Status m_threadStartState;
bool m_isStreaming;
unsigned int m_buffers[BufferCount];
unsigned int m_channelCount;
unsigned int m_sampleRate;
Uint32 m_format;
bool m_loop;
Uint64 m_samplesProcessed;
bool m_endBuffers[BufferCount];

} // namespace sf

#ifdef // SFML_SOUNDSTREAM_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Sprite.hpp

```cpp
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include <SFML/Graphics/Vertex.hpp>
#include <SFML/Graphics/Rect.hpp>
```
namespace sf
{
    class Texture;

    class SFML_GRAPHICS_API Sprite : public Drawable, public Transformable
    {
    public:
        Sprite();
        explicit Sprite(const Texture& texture);
        Sprite(const Texture& texture, const IntRect& rectangle);
        void setTexture(const Texture& texture, bool resetRect = false);
        void setTextureRect(const IntRect& rectangle);
        void setColor(const Color& color);
        const Texture* getTexture() const;
        const IntRect& getTextureRect() const;
        const Color& getColor() const;
        FloatRect getLocalBounds() const;
        FloatRect getGlobalBounds() const;
    private:
        virtual void draw(RenderTarget& target, RenderStates states);
        void updatePositions();
        void updateTexCoords();
        // Member data
        Vertex m_vertices[4];
        const Texture* m_texture;
        IntRect m_textureRect;
    } // namespace sf

#if !SFML_SPRITE_HPP
    #endif // SFML_SPRITE_HPP

// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_STRING_HPP
#define SFML_STRING_HPP

// Headers
#include <SFML/System/Export.hpp>
#include <SFML/System/Utf.hpp>
#include <iterator>
#include <locale>
#include <string>
#endif // SFML_STRING_HPP
namespace sf
{
    class SFML_SYSTEM_API String
    {
    public:

        // Types
        typedef std::basic_string<Uint32>::iterator Iterator;
        typedef std::basic_string<Uint32>::const_iterator ConstIterator;

        // Static member data
        static const std::size_t InvalidPos;

        String();

        String(char ansiChar, const std::locale& locale = std::locale());
        String(wchar_t wideChar);
        String(Uint32 utf32Char);
        String(const char* ansiString, const std::locale& locale = std::locale());
        String(const std::string& ansiString, const std::locale& locale = std::locale());
        String(const wchar_t* wideString);
        String(const std::wstring& wideString);
        String(const Uint32* utf32String);
        String(const std::basic_string<Uint32>& utf32String);
        String(const String& copy);

        template <typename T>
        static String fromUtf8(T begin, T end);

        template <typename T>
        static String fromUtf16(T begin, T end);

        template <typename T>
        static String fromUtf32(T begin, T end);

        operator std::string() const;
        operator std::wstring() const;

        std::string toAnsiString(const std::locale& locale = std::locale());
std::wstring toString() const;
std::basic_string<Uint8> toUtf8() const;
std::basic_string<Uint16> toUtf16() const;
std::basic_string<Uint32> toUtf32() const;

String& operator =(const String& right);
String& operator +=(const String& right);
Uint32 operator [](std::size_t index) const;
Uint32& operator [](std::size_t index);

void clear();
std::size_t getSize() const;
bool isEmpty() const;

void erase(std::size_t position, std::size_t count = 1);
void insert(std::size_t position, const String& str);
std::size_t find(const String& str, std::size_t start = 0) const;
void replace(std::size_t position, std::size_t length, const String& str);
void replace(const String& searchFor, const String& replaceWith);

String substring(std::size_t position, std::size_t length = InvalidPos) const;

const Uint32* getData() const;

Iterator begin();
ConstIterator begin() const;
Iterator end();
ConstIterator end() const;

private:
friend SFML_SYSTEM_API bool operator ==(const String& left, const String& right);
friend SFML_SYSTEM_API bool operator <(const String& left, const String& right);

// Member data
std::basic_string<Uint32> m_string;
SFML_SYSTEM_API bool operator ==(const String& left, const String& right)
SFML_SYSTEM_API bool operator !=(const String& left, const String& right)
SFML_SYSTEM_API bool operator <(const String& left, const String& right)
SFML_SYSTEM_API bool operator >(const String& left, const String& right)
SFML_SYSTEM_API bool operator <=(const String& left, const String& right)
SFML_SYSTEM_API bool operator >=(const String& left, const String& right)
SFML_SYSTEM_API String operator +(const String& left, const String& right)

#include <SFML/System/String.inl>

} // namespace sf

#endif // SFML_STRING_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
System.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty.  In no event will the authors be held liable for any
7 // damages arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and
11 // redistribute it freely, subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 //    If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is
17 //    not required.
18 //
19 // 2. Altered source versions must be plainly marked as such,
20 //    and must not be misrepresented as being the original software.
21 //
22 // 3. This notice may not be removed or altered from any source
23 //    code.
24 //
25 #ifndef SFML_SYSTEM_HPP
26 #define SFML_SYSTEM_HPP
27
28 // Headers
29 #include <SFML/Config.hpp>
30 #include <SFML/System/Clock.hpp>
31 #include <SFML/System/Err.hpp>
32 #include <SFML/System/FileInputStream.hpp>
TcpListener.hpp

```cpp
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty. In no event will the authors be held liable for any damages
7 // arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and
11 // redistribute it freely, subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 //    If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //    required.
18 //
19 // 2. Altered source versions must be plainly marked as such,
20 //    and must not be misrepresented as being the original software.
21 //
22 // 3. This notice may not be removed or altered from any source
23 //
24 #ifndef SFML_TCPLISTENER_HPP
25 #define SFML_TCPLISTENER_HPP
26 #define SFML_TCPLISTENER_HPP
27 // Headers
28 #include <SFML/Network/Export.hpp>
29 #include <SFML/Network/Socket.hpp>
30 #include <SFML/Network/IpAddress.hpp>
31 #include <SFML/Network/Export.hpp>
32 #include <SFML/Network/Socket.hpp>
33 #include <SFML/Network/IpAddress.hpp>
34 35
namespace sf
{
    class TcpSocket;
}

class SFML_NETWORK_API TcpListener : public Socket
{
public:
    TcpListener();
    unsigned short getLocalPort() const;
    Status listen(unsigned short port, const IpAddress& address);
    void close();
    Status accept(TcpSocket& socket);
};

} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_TCPSOCKET_HPP
#define SFML_TCPSOCKET_HPP

// Headers
#include <SFML/Network/Export.hpp>
#include <SFML/Network/Socket.hpp>
#include <SFML/System/Time.hpp>
#endif


namespace sf
{
  class TcpListener;
  class IpAddress;
  class Packet;

  class SFML_NETWORK_API TcpSocket : public Socket
  {
    public:

    TcpSocket();

    unsigned short getLocalPort() const;

    IpAddress getRemoteAddress() const;

    unsigned short getRemotePort() const;

    Status connect(const IpAddress& remoteAddress, unsigned short
                    Time::Zero);

    void disconnect();

    Status send(const void* data, std::size_t size);

    Status send(const void* data, std::size_t size, std::size_t& sent);

    Status receive(void* data, std::size_t size, std::size_t& received);

    Status send(Packet& packet);

    Status receive(Packet& packet);

  private:

    friend class TcpListener;

    struct PendingPacket
    {
      PendingPacket();

      Uint32 Size;
      std::size_t SizeReceived;
      std::vector<char> Data;
    };

    // Member data
    PendingPacket m_pendingPacket;

  } // namespace sf
SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen.
Text.hpp

1  //
2  // SFML - Simple and Fast Multimedia Library
3  // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4  //
5  // This software is provided 'as-is', without any express or implied
6  // warranty.  In no event will the authors be held liable for any damages
7  // arising from the use of this software.
8  //
9  // Permission is granted to anyone to use this software for any
10  // purpose, including commercial applications, and to alter it and redist-
11  // tribute it freely, subject to the following restrictions:
12  //
13  // 1. The origin of this software must not be misrepresented;
14  //    you must not claim that you wrote the original software.
15  //    If you use this software in a product, an acknowledgment
16  //    in the product documentation would be appreciated but is not
17  //    required.
18  //
19  // 2. Altered source versions must be plainly marked as such, and
20  //    must not be misrepresented as being the original software.
21  //
22  // 3. This notice may not be removed or altered from any source
23  //
24  #ifndef SFML_TEXT_HPP
25  #define SFML_TEXT_HPP
26  // Headers
27  #include <SFML/Graphics/Export.hpp>
28  #include <SFML/Graphics/Drawable.hpp>
29  #include <SFML/Graphics/Transformable.hpp>
30  #include <SFML/Graphics/Font.hpp>
31  #include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/VertexArray.hpp>
#include <SFML/System/String.hpp>
#include <string>
#include <vector>

namespace sf
{
    class SFML_GRAPHICS_API Text : public Drawable, public Transformable
    {
    public:
        enum Style
        {
            Regular = 0,
            Bold = 1 << 0,
            Italic = 1 << 1,
            Underlined = 1 << 2,
            StrikeThrough = 1 << 3
        };

        Text();

        Text(const String& string, const Font& font, unsigned int characterSize);

        void setString(const String& string);
        void setFont(const Font& font);
        void setCharacterSize(unsigned int size);
        void setStyle(Uint32 style);

        SFML_DEPRECATED void setColor(const Color& color);
        void setFillColor(const Color& color);
        void setOutlineColor(const Color& color);
        void setOutlineThickness(float thickness);

        const String& getString() const;
        const Font* getFont() const;
        unsigned int getCharacterSize() const;
        Uint32 getStyle() const;

        SFML_DEPRECATED const Color& getColor() const;

        const Color& getFillColor() const;
    };
}
const Color& getOutlineColor() const;
float getOutlineThickness() const;
Vector2f findCharacterPos(std::size_t index) const;
FloatRect getLocalBounds() const;
FloatRect getGlobalBounds() const;

private:
virtual void draw(RenderTarget& target, RenderStates states);
void ensureGeometryUpdate() const;

// Member data
String m_string;
const Font* m_font;
unsigned int m_characterSize;
Uint32 m_style;
Color m_fillColor;
Color m_outlineColor;
float m_outlineThickness;
mutable VertexArray m_vertices;
mutable VertexArray m_outlineVertices;
mutable FloatRect m_bounds;
mutable bool m_geometryNeedUpdate;

};

// namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
Texture.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty.  In no event will the authors be held liable for any damages
7 // arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redist
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 //    If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //    required.
18 //
19 // 2. Altered source versions must be plainly marked as such, and
20 //    must not be misrepresented as being the original software.
21 //
22 // 3. This notice may not be removed or altered from any source
23 //
24 #ifndef SFML_TEXTURE_HPP
25 #define SFML_TEXTURE_HPP
26 #define SFML_TEXTURE_HPP
27 //
28 // Headers
29 #include <SFML/Graphics/Export.hpp>
30 #include <SFML/Graphics/Image.hpp>
31 #include <SFML/Graphics/Image.hpp>
32 //include <SFML/Graphics/Image.hpp>
33 #include <SFML/Window/GlResource.hpp>
34 35
namespace sf {
  class Window;
  class RenderTarget;
  class RenderTexture;
  class InputStream;

  class SFML_GRAPHICS_API Texture : GlResource {
    public:
      enum CoordinateType {
        Normalized,
        Pixels
      };

    public:
      Texture();
      Texture(const Texture& copy);
      ~Texture();

      bool create(unsigned int width, unsigned int height);
      bool loadFromFile(const std::string& filename, const IntRect& area);
      bool loadFromMemory(const void* data, std::size_t size, const IntRect& area);
      bool loadFromStream(InputStream& stream, const IntRect& area);
      bool loadFromImage(const Image& image, const IntRect& area);

      Vector2u getSize() const;
      Image copyToImage() const;

      void update(const Uint8* pixels);
      void update(const Uint8* pixels, unsigned int width, unsigned int height);
      void update(const Uint8* pixels, unsigned int width, unsigned int height, unsigned int y);
      void update(const Image& image);
      void update(const Image& image, unsigned int x, unsigned int y);
      void update(const Window& window);
      void update(const Window& window, unsigned int x, unsigned int y);
  }
}
void setSmooth(bool smooth);

bool isSmooth() const;

void setSrgb(bool sRgb);

bool isSrgb() const;

void setRepeated(bool repeated);

bool isRepeated() const;

bool generateMipmap();

Texture& operator =(const Texture& right);

unsigned int getNativeHandle() const;

static void bind(const Texture* texture, CoordinateType coordinateType = Normalized);

static unsigned int getMaximumSize();

private:

friend class RenderTexture;
friend class RenderTarget;

static unsigned int getValidSize(unsigned int size);

void invalidateMipmap();

// Member data
Vector2u m_size;
Vector2u m_actualSize;
unsigned int m_texture;
bool m_isSmooth;
bool m_sRgb;
bool m_isRepeated;
mutable bool m_pixelsFlipped;
bool m_fboAttachment;
bool m_hasMipmap;
Uint64 m_cacheId;

} // namespace sf

#endif // SFML_TEXTURE_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications,
// and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment
//    in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_THREAD_HPP
#define SFML_THREAD_HPP

// Headers
#include <SFML/System/Export.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <cstdlib>

#endif
namespace sf
{
namespace priv
{
class ThreadImpl;
 struct ThreadFunc;
}

class SFML_SYSTEM_API Thread : NonCopyable
{
public:

template <typename F>
    Thread(F function);

template <typename F, typename A>
    Thread(F function, A argument);

template <typename C>
    Thread(void(C::*function)(), C* object);

    ~Thread();

void launch();
void wait();
void terminate();

private:

friend class priv::ThreadImpl;

void run();

// Member data
priv::ThreadImpl* m_impl;
priv::ThreadFunc* m_entryPoint;

};

#include <SFML/System/Thread.inl>

} // namespace sf

} // namespace sf

SFML is licensed under the terms and conditions of the zlib/png license
# ThreadLocal.hpp

```cpp
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty. In no event will the authors be held liable for any damages
7 // arising from the use of this software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // purpose, including commercial applications, and to alter it and redist
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 //    you must not claim that you wrote the original software.
15 // 2. If you use this software in a product, an acknowledgment
16 //    in the product documentation would be appreciated but is not
17 //    required.
18 // 3. Altered source versions must be plainly marked as such,
19 //    and must not be misrepresented as being the original software.
20 //
21 // This notice may not be removed or altered from any source
22 //
23
24 #ifndef SFML_THREADLOCAL_HPP
25 #define SFML_THREADLOCAL_HPP
26 #define SFML_THREADLOCAL_HPP
27
28 // Headers
29 #include <SFML/System/Export.hpp>
30 #include <SFML/System/NonCopyable.hpp>
31 #include <cstdlib>
32 #include <cstddef>
```
namespace sf
{
    namespace priv
    {
        class ThreadLocalImpl;
    }

class SFML_SYSTEM_API ThreadLocal : NonCopyable
{
public:
    ThreadLocal(void* value = NULL);
    ~ThreadLocal();
    void setValue(void* value);
    void* getValue() const;

private:
    // Member data
    priv::ThreadLocalImpl* m_impl;
};

} // namespace sf

#endif // SFML_THREADLOCAL_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source.

#ifndef SFML_THREADLOCALPTR_HPP
#define SFML_THREADLOCALPTR_HPP

namespace sf {

// Headers
#include "SFML/System/ThreadLocal.hpp"

namespace sf {

template <typename T>
class ThreadLocalPtr : private ThreadLocal
{
public:
    ThreadLocalPtr(T* value = NULL);
    T& operator *() const;
    T* operator ->() const;
    operator T*() const;
    ThreadLocalPtr<T>& operator =(T* value);
    ThreadLocalPtr<T>& operator =(const ThreadLocalPtr<T>& right);
};

#include <SFML/System/ThreadLocalPtr.inl>

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila. :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_TIME_HPP
#define SFML_TIME_HPP

// Headers
#include <SFML/System/Export.hpp>

namespace sf {

class SFML_SYSTEM_API Time
{
public:
    Time();
    float asSeconds() const;
    Int32 asMilliseconds() const;
    Int64 asMicroseconds() const;

    // Static member data
    static const Time Zero;

private:
    friend SFML_SYSTEM_API Time seconds(float);
    friend SFML_SYSTEM_API Time milliseconds(Int32);
    friend SFML_SYSTEM_API Time microseconds(Int64);

    explicit Time(Int64 microseconds);

    // Member data
    Int64 m_microseconds;
};

SFML_SYSTEM_API Time seconds(float amount);
SFML_SYSTEM_API Time milliseconds(Int32 amount);
SFML_SYSTEM_API Time microseconds(Int64 amount);
SFML_SYSTEM_API bool operator ==(Time left, Time right);
SFML_SYSTEM_API bool operator !(Time left, Time right);
SFML_SYSTEM_API bool operator <(Time left, Time right);
SFML_SYSTEM_API bool operator >(Time left, Time right);
SFML_SYSTEM_API bool operator <=(Time left, Time right);
SFML_SYSTEM_API bool operator >=(Time left, Time right);
SFML_SYSTEM_API Time operator -(Time right);
SFML_SYSTEM_API Time operator +(Time left, Time right);
SFML_SYSTEM_API Time& operator +=(Time& left, Time right);
SFML_SYSTEM_API Time operator -(Time left, Time right);
SFML_SYSTEM_API Time& operator -=(Time& left, Time right);
SFML_SYSTEM_API Time operator *(Time left, float right);
SFML_SYSTEM_API Time operator *(Time left, Int64 right);
SFML_SYSTEM_API Time operator *(float left, Time right);
SFML_SYSTEM_API Time operator *(Int64 left, Time right);
SFML_SYSTEM_API Time& operator *=(Time& left, float right);
SFML_SYSTEM_API Time& operator *=(Time& left, Int64 right);
SFML_SYSTEM_API Time operator /(Time left, float right);
SFML_SYSTEM_API Time operator /(Time left, Int64 right);
SFML_SYSTEM_API Time& operator /= (Time& left, float right);
SFML_SYSTEM_API Time& operator /= (Time& left, Int64 right);
SFML_SYSTEM_API float operator /(Time left, Time right);
SFML_SYSTEM_API Time operator %(Time left, Time right);
SFML_SYSTEM_API Time& operator %= (Time& left, Time right);

// namespace sf

#if !defined(SFML_TIME_HPP)

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen

#endif // SFML_TIME_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied
// warranty.  In no event will the authors be held liable for any damages
// arising from the use of this software.
//
// Permission is granted to anyone to use this software for any
// purpose, including commercial applications, and to alter it and redistri
// bution subject to the following restrictions:
// 1. The origin of this software must not be misrepresented;
// 2. Altered source versions must be plainly marked as such, and must not
//    be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source

#ifndef SFML_TOUCH_HPP
#define SFML TOUCH_HPP

// Headers
#include <SFML/Window/Export.hpp>
#include <SFML/System/Vector2.hpp>

namespace sf {
```cpp
36 {  
37   class Window;
38
43   class SFML_WINDOW_API Touch {
44     public:
45       static bool isDown(unsigned int finger);
46       static Vector2i getPosition(unsigned int finger);
47       static Vector2i getPosition(unsigned int finger, const Window&);
48     };
49   } // namespace sf
50
51 #endif // SFML_TOUCH_HPP
```

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment
//    in the product documentation would be appreciated but is not
//    required.

// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source

#ifndef SFML_TRANSFORM_HPP
#define SFML_TRANSFORM_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/System/Vector2.hpp>

#endif
namespace sf
{
    class SFML_GRAPHICS_API Transform
    {
    public:
        Transform();
        Transform(float a00, float a01, float a02,
                  float a10, float a11, float a12,
                  float a20, float a21, float a22);
        const float* getMatrix() const;
        Transform getInverse() const;
        Vector2f transformPoint(float x, float y) const;
        Vector2f transformPoint(const Vector2f& point) const;
        FloatRect transformRect(const FloatRect& rectangle) const;
        Transform& combine(const Transform& transform);
        Transform& translate(float x, float y);
        Transform& translate(const Vector2f& offset);
        Transform& rotate(float angle);
        Transform& rotate(float angle, float centerX, float centerY);
        Transform& rotate(float angle, const Vector2f& center);
        Transform& scale(float scaleX, float scaleY);
        Transform& scale(float scaleX, float scaleY, float centerX, float centerY);
        Transform& scale(const Vector2f& factors);
        Transform& scale(const Vector2f& factors, const Vector2f& center);
        // Static member data
        static const Transform Identity;
    private:
        // Member data
        float m_matrix[16];
    }
}
SFML_GRAPHICS_API Transform operator *(const Transform& left, const
SFML_GRAPHICS_API Transform& operator *=(Transform& left, const Vector2f& right);
Transformable.hpp

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // warranty.  In no event will the authors be held liable for any damages a
7 // software.
8 //
9 // Permission is granted to anyone to use this software for any
10 // including commercial applications, and to alter it and redist
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 // you must not claim that you wrote the original software.
15 // If you use this software in a product, an acknowledgment
16 // in the product documentation would be appreciated but is not
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 // and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source
22 //
23 #ifndef SFML_TRANSFORMABLE_HPP
24 #define SFML_TRANSFORMABLE_HPP
25 #endif
26
27 // Headers
28 #include <SFML/Graphics/Export.hpp>
29 #include <SFML/Graphics/Transform.hpp>
30
31 namespace sf
```cpp
class SFML_GRAPHICS_API Transformable public:

Transformable();

virtual ~Transformable();

void setPosition(float x, float y);

void setPosition(const Vector2f& position);

void setRotation(float angle);

void setScale(float factorX, float factorY);

void setScale(const Vector2f& factors);

void setOrigin(float x, float y);

void setOrigin(const Vector2f& origin);

const Vector2f& getPosition() const;

float getRotation() const;

const Vector2f& getScale() const;

const Vector2f& getOrigin() const;

void move(float offsetX, float offsetY);

void move(const Vector2f& offset);

void rotate(float angle);

void scale(float factorX, float factorY);

void scale(const Vector2f& factor);

const Transform& getTransform() const;

const Transform& getInverseTransform() const;

private:

    // Member data
    Vector2f m_origin;
    Vector2f m_position;
    float m_rotation;
    Vector2f m_scale;
```
mutable Transform m_transform;
mutable bool m_transformNeedUpdate;
mutable Transform m_inverseTransform;
mutable bool m_inverseTransformNeedUpdate;
};

} // namespace sf

#endif	//	SFML_TRANSFORMABLE_HPP
_sfml_2.4.2

UdpSocket.hpp

```
1 //
3 // SFML - Simple and Fast Multimedia Library
4 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
5 //
6 // This software is provided 'as-is', without any express or imp
7 // In no event will the authors be held liable for any damages a
8 // Permission is granted to anyone to use this software for any
9 // including commercial applications, and to alter it and redist
10 // subject to the following restrictions:
11 //
12 // 1. The origin of this software must not be misrepresented;
13 // you must not claim that you wrote the original software.
14 // If you use this software in a product, an acknowledgment
15 // in the product documentation would be appreciated but is n
16 // 2. Altered source versions must be plainly marked as such,
17 // and must not be misrepresented as being the original softw
18 //
19 // 3. This notice may not be removed or altered from any sourc
20 //
21 #ifndef SFML_UDPSOCKET_HPP
22 #define SFML_UDPSOCKET_HPP
23
24 // Headers
25 #include <SFML/Network/Export.hpp>
26 #include <SFML/Network/Socket.hpp>
27 #include <SFML/Network/IpAddress.hpp>
28 #include <vector>
29```
namespace sf
{
    class Packet;
}

class SFML_NETWORK_API UdpSocket : public Socket
{
public:
    // Constants
    enum {
        MaxDatagramSize = 65507
    };

    UdpSocket();
    unsigned short getLocalPort() const;
    Status bind(unsigned short port, const IpAddress& address =);
    void unbind();
    Status send(const void* data, std::size_t size, const IpAddress& remotePort);
    Status receive(void* data, std::size_t size, std::size_t& received, IpAddress& remoteAddress, unsigned short& remotePort);
    Status send(Packet& packet, const IpAddress& remoteAddress,);
    Status receive(Packet& packet, IpAddress& remoteAddress, unsigned);
private:
    // Member data
    std::vector<char> m_buffer;
};

}; // namespace sf

#endif // SFML_UDPSOCKET_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_UTF_HPP
#define SFML_UTF_HPP

// Headers
#include <SFML/Config.hpp>
#include <algorithm>
#include <locale>
#include <string>
#include <cstdlib>

#endif // SFML_UTF_HPP
namespace sf {

template <unsigned int N>
class Utf;

template <>
class Utf<8> {
public:

    template <typename In>
    static In decode(In begin, In end, Uint32& output, Uint32 replacement = 0);

    template <typename Out>
    static Out encode(Uint32 input, Out output, Uint8 replacement = 0);

    template <typename In>
    static In next(In begin, In end);

    template <typename In>
    static std::size_t count(In begin, In end);

    template <typename In, typename Out>
    static Out fromAnsi(In begin, In end, Out output, const std::locale& locale = std::locale());

    template <typename In, typename Out>
    static Out fromWide(In begin, In end, Out output);

    template <typename In, typename Out>
    static Out fromLatin1(In begin, In end, Out output);        

    template <typename In, typename Out>
    static Out toAnsi(In begin, In end, Out output, char replacement = 0, locale = std::locale());

    template <typename In, typename Out>
    static Out toWide(In begin, In end, Out output, wchar_t replacement = 0);

    template <typename In, typename Out>
    static Out toLatin1(In begin, In end, Out output, char replacement = 0);

    template <typename In, typename Out>
    static Out toUtf8(In begin, In end, Out output);

    template <typename In, typename Out>
    static Out toUtf16(In begin, In end, Out output);

    template <typename In, typename Out>
    static Out toUtf32(In begin, In end, Out output);
template <>
class Utf<16> {
public:
    template <typename In>
    static In decode(In begin, In end, Uint32& output, Uint32 replacement = 0);
    template <typename Out>
    static Out encode(Uint32 input, Out output, Uint16 replacement = 0);
    template <typename In>
    static In next(In begin, In end);
    template <typename In>
    static std::size_t count(In begin, In end);
    template <typename In, typename Out>
    static Out fromAnsi(In begin, In end, Out output, const std::locale& locale = std::locale());
    template <typename In, typename Out>
    static Out fromWide(In begin, In end, Out output);
    template <typename In, typename Out>
    static Out fromLatin1(In begin, In end, Out output);
    template <typename In, typename Out>
    static Out toAnsi(In begin, In end, Out output, char replacement = 0, locale = std::locale());
    template <typename In, typename Out>
    static Out toWide(In begin, In end, Out output, wchar_t replacement = 0);
    template <typename In, typename Out>
    static Out toLatin1(In begin, In end, Out output, char replacement = 0);
    template <typename In, typename Out>
    static Out toUtf8(In begin, In end, Out output);
    template <typename In, typename Out>
    static Out toUtf16(In begin, In end, Out output);
    template <typename In, typename Out>
    static Out toUtf32(In begin, In end, Out output);
};

template <>
class Utf<32> {
}
public:

template <typename In>
static In decode(In begin, In end, Uint32& output, Uint32 replacement = 0);

template <typename Out>
static Out encode(Uint32 input, Out output, Uint32 replacement = 0);

template <typename In>
static In next(In begin, In end);

template <typename In>
static std::size_t count(In begin, In end);

template <typename In, typename Out>
static Out fromAnsi(In begin, In end, Out output, const std::locale& locale = std::locale());

template <typename In, typename Out>
static Out fromWide(In begin, In end, Out output);

template <typename In, typename Out>
static Out fromLatin1(In begin, In end, Out output);

template <typename In, typename Out>
static Out toAnsi(In begin, In end, Out output, char replacement = 0, std::locale& locale = std::locale());

template <typename In, typename Out>
static Out toWide(In begin, In end, Out output, wchar_t replacement = 0);

template <typename In, typename Out>
static Out toLatin1(In begin, In end, Out output, char replacement = 0);

template <typename In, typename Out>
static Out toUtf8(In begin, In end, Out output);

template <typename In, typename Out>
static Out toUtf16(In begin, In end, Out output);

template <typename In, typename Out>
static Out toUtf32(In begin, In end, Out output);

template <typename In>
static Uint32 decodeAnsi(In input, const std::locale& locale);

template <typename In>
static Uint32 decodeWide(In input);

template <typename Out>
static Out encodeAnsi(Uint32 codepoint, Out output, char replacement = 0, std::locale& locale = std::locale());
template <typename Out>
static Out encodeWide(Uint32 codepoint, Out output, wchar_t replacement = 0);

#include <SFML/System/Utf.inl>

// Make typedefs to get rid of the template syntax
typedef Utf<8> Utf8;
typedef Utf<16> Utf16;
typedef Utf<32> Utf32;

} // namespace sf

#ifndef SFML_UTF_HPP
#endif // SFML_UTF_HPP
Vector2.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_VECTOR2_HPP
#define SFML_VECTOR2_HPP

namespace sf {

    template<typename T>
    class Vector2 {

    }

}
#endif // SFML_VECTOR2_HPP
```
public:
    Vector2();
    Vector2(T X, T Y);

    // Member data
    T x;
    T y;
};

template <typename U>
explicit Vector2(const Vector2<U>& vector);

#include <SFML/System/Vector2.inl>
// Define the most common types
typedef Vector2<int> Vector2i;
typedef Vector2<unsigned int> Vector2u;
typedef Vector2<float> Vector2f;
}
} // namespace sf

#endif // SFML_VECTOR2_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source version.

#ifndef SFML_VECTOR3_HPP
#define SFML_VECTOR3_HPP

namespace sf
{

    template<typename T>
    class Vector3
    {

    public:

    private:

    protected:

    

} // Vector3

#endif // SFML_VECTOR3_HPP
public:
    Vector3();
    Vector3(T X, T Y, T Z);

    template <typename U>
    explicit Vector3(const Vector3<U>& vector);

    // Member data
    T x;
    T y;
    T z;
    
    template <typename T>
    Vector3<T> operator -(const Vector3<T>& left);

    template <typename T>
    Vector3<T>& operator +=(Vector3<T>& left, const Vector3<T>& right);

    template <typename T>
    Vector3<T>& operator -=(Vector3<T>& left, const Vector3<T>& right);

    template <typename T>
    Vector3<T> operator +(const Vector3<T>& left, const Vector3<T>& right);

    template <typename T>
    Vector3<T> operator -(const Vector3<T>& left, const Vector3<T>& right);

    template <typename T>
    Vector3<T> operator *(T left, const Vector3<T>& right);

    template <typename T>
    Vector3<T>& operator *= (Vector3<T>& left, T right);

    template <typename T>
    Vector3<T> operator /(const Vector3<T>& left, T right);

    template <typename T>
    Vector3<T>& operator /= (Vector3<T>& left, T right);

    template <typename T>
    bool operator ==(const Vector3<T>& left, const Vector3<T>& right);

    template <typename T>
    bool operator !=(const Vector3<T>& left, const Vector3<T>& right);

#include <SFML/System/Vector3.inl>
// Define the most common types
typedef Vector3<int> Vector3i;
typedef Vector3<float> Vector3f;

} // namespace sf

#define // SFML_VECTOR3_HPP

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_VERTEX_HPP
#define SFML_VERTEX_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Color.hpp>
#include <SFML/System/Vector2.hpp>

#endif
namespace sf {

class SFML_GRAPHICS_API Vertex {

public:

    Vertex();
    Vertex(const Vector2f& thePosition);
    Vertex(const Vector2f& thePosition, const Color& theColor);
    Vertex(const Vector2f& thePosition, const Vector2f& theTexCoords);
    Vertex(const Vector2f& thePosition, const Color& theColor, const Vector2f& texCoords);

    // Member data
    Vector2f position;
    Color color;
    Vector2f texCoords;
};

} // namespace sf

//SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
#ifndef SFML_VERTEXARRAY_HPP
#define SFML_VERTEXARRAY_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Vertex.hpp>
#include <SFML/Graphics/PrimitiveType.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/Drawable.hpp>
```cpp
#include <vector>

namespace sf
{
    class SFML_GRAPHICS_API VertexArray : public Drawable
    {
    public:
        VertexArray();
        explicit VertexArray(PrimitiveType type, std::size_t vertexCount);
        std::size_t getVertexCount() const;
        Vertex& operator [](std::size_t index);
        const Vertex& operator [](std::size_t index) const;
        void clear();
        void resize(std::size_t vertexCount);
        void append(const Vertex& vertex);
        void setPrimitiveType(PrimitiveType type);
        PrimitiveType getPrimitiveType() const;
        FloatRect getBounds() const;
    private:
        virtual void draw(RenderTarget& target, RenderStates states);
    private:
        // Member data
        std::vector<Vertex> m_vertices;
        PrimitiveType m_primitiveType;
    };
}

// namespace sf

#include <SFML_VERTEXARRAY_HPP>
```
VideoMode.hpp

```
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//   you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such,
//   and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_VIDEOMODE_HPP
#define SFML_VIDEOMODE_HPP

// Headers
#include <SFML/Window/Export.hpp>
#include <vector>

namespace sf
```
class SFML_WINDOW_API VideoMode
{
    public:
    VideoMode();
    VideoMode(unsigned int modeWidth, unsigned int modeHeight, unsigned int bitsPerPixel);

    static VideoMode getDesktopMode();
    static const std::vector<VideoMode>& getFullscreenModes();
    bool isValid() const;

    // Member data
    unsigned int width;
    unsigned int height;
    unsigned int bitsPerPixel;
};

SFML_WINDOW_API bool operator ==(const VideoMode& left, const VideoMode& right);
SFML_WINDOW_API bool operator !=(const VideoMode& left, const VideoMode& right);
SFML_WINDOW_API bool operator <(const VideoMode& left, const VideoMode& right);
SFML_WINDOW_API bool operator >(const VideoMode& left, const VideoMode& right);
SFML_WINDOW_API bool operator <=(const VideoMode& left, const VideoMode& right);
SFML_WINDOW_API bool operator >=(const VideoMode& left, const VideoMode& right);

} // namespace sf

#endif // SFML_VIDEOMODE_HPP
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source.

#ifndef SFML_VIEW_HPP
#define SFML_VIEW_HPP

// Headers
#include <SFML/Graphics/Export.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/Transform.hpp>
#include <SFML/System/Vector2.hpp>
#endif
namespace sf {

class SFML_GRAPHICS_API View {

public:

    View();

    explicit View(const FloatRect& rectangle);

    View(const Vector2f& center, const Vector2f& size);

    void setCenter(float x, float y);

    void setCenter(const Vector2f& center);

    void setSize(float width, float height);

    void setSize(const Vector2f& size);

    void setRotation(float angle);

    void setViewport(const FloatRect& viewport);

    void reset(const FloatRect& rectangle);

    const Vector2f& getCenter() const;

    const Vector2f& getSize() const;

    float getRotation() const;

    const FloatRect& getViewport() const;

    void move(float offsetX, float offsetY);

    void move(const Vector2f& offset);

    void rotate(float angle);

    void zoom(float factor);

    const Transform& getTransform() const;

    const Transform& getInverseTransform() const;

private:

    // Member data
    Vector2f m_center;
    Vector2f m_size;
float m_rotation;
FloatRect m_viewport;
mutable Transform m_transform;
mutable Transform m_inverseTransform;
mutable bool m_transformUpdated;
mutable bool m_invTransformUpdated;
}
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:

// 1. The origin of this software must not be misrepresented;
//    you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment
//    in the product documentation would be appreciated but is not
// 2. Altered source versions must be plainly marked as such,
//    and must not be misrepresented as being the original software.
// 3. This notice may not be removed or altered from any source

#ifndef SFML_WINDOW_HPP
#define SFML_WINDOW_HPP

// Headers
#include <SFML/Window/Export.hpp>
#include <SFML/Window/ContextSettings.hpp>
#include <SFML/Window/VideoMode.hpp>
#include <SFML/Window/WindowHandle.hpp>
#include <SFML/Window/WindowStyle.hpp>
#endif
```cpp
#include <SFML/Window/GlResource.hpp>
#include <SFML/System/Clock.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <SFML/System/String.hpp>

namespace sf
{
  namespace priv
  {
    class GlContext;
    class WindowImpl;
  }

class Event;

class SFML_WINDOW_API Window : GlResource, NonCopyable
{
public:
  Window();

  Window(VideoMode mode, const String& title, Uint32 style = ContextSettings& settings = ContextSettings());

  explicit Window(WindowHandle handle, const ContextSettings& settings = ContextSettings());

  virtual ~Window();

  void create(VideoMode mode, const String& title, Uint32 style = ContextSettings& settings = ContextSettings());

  void create(WindowHandle handle, const ContextSettings& settings = ContextSettings());

  void close();

  bool isOpen() const;

  const ContextSettings& getSettings() const;

  bool pollEvent(Event& event);

  bool waitEvent(Event& event);

  Vector2i getPosition() const;

  void setPosition(const Vector2i& position);

  Vector2u getSize() const;

  void setSize(const Vector2u& size);
};
```
```cpp
void setTitle(const String& title);
void setIcon(unsigned int width, unsigned int height, const
void setVerticalSyncEnabled(bool enabled);
void setMouseMoveCursorVisible(bool visible);
void setMouseMoveCursorGrabbed(bool grabbed);
void setKeyRepeatEnabled(bool enabled);
void setFramerateLimit(unsigned int limit);
bool setActive(bool active = true) const;
void requestFocus();
bool hasFocus() const;
void display();
WindowHandle getSystemHandle() const;
protected:
virtual void onCreate();
virtual void onResize();
private:
bool filterEvent(const Event& event);
void initialize();
// Member data
priv::WindowImpl* m_impl;
priv::GlContext* m_context;
Clock m_clock;
Time m_frameTimeLimit;
Vector2u m_size;
};
```
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
Window.hpp

```cpp
// SFML - Simple and Fast Multimedia Library
// Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)

// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.

// Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

// 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
//    If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

// 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

// 3. This notice may not be removed or altered from any source distribution.

#ifndef SFML_SFML_WINDOW_HPP
#define SFML_SFML_WINDOW_HPP

// Headers
#include <SFML/System.hpp>
#include <SFML/Window/Context.hpp>
#include <SFML/Window/ContextSettings.hpp>
#include <SFML/Window/Event.hpp>
#endif
```
```cpp
#include <SFML/Window/Joystick.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <SFML/Window/Mouse.hpp>
#include <SFML/Window/Sensor.hpp>
#include <SFML/Window/Touch.hpp>
#include <SFML/Window/VideoMode.hpp>
#include <SFML/Window/Window.hpp>
#include <SFML/Window/WindowHandle.hpp>
#include <SFML/Window/WindowStyle.hpp>

#include <SFML/Window/Joystick.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <SFML/Window/Mouse.hpp>
#include <SFML/Window/Sensor.hpp>
#include <SFML/Window/Touch.hpp>
#include <SFML/Window/VideoMode.hpp>
#include <SFML/Window/Window.hpp>
#include <SFML/Window/WindowHandle.hpp>
#include <SFML/Window/WindowStyle.hpp>

#endif // SFML_SFML_WINDOW_HPP
```
WindowHandle.hpp

```cpp
1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
4 //
5 // This software is provided 'as-is', without any express or implied
6 // In no event will the authors be held liable for any damages arising
7 // Permission is granted to anyone to use this software for any
8 // subject to the following restrictions:
9 // 1. The origin of this software must not be misrepresented;
10 // you must not claim that you wrote the original software.
11 // If you use this software in a product, an acknowledgment
12 // in the product documentation would be appreciated but is not
13 // 2. Altered source versions must be plainly marked as such,
14 // and must not be misrepresented as being the original software.
15 // 3. This notice may not be removed or altered from any source
16 //
17 #ifndef SFML_WINDOWHANDLE_HPP
18 #define SFML_WINDOWHANDLE_HPP
19 #include <SFML/Config.hpp>
20 // Headers
21 #include <SFML/Config.hpp>
22 // Windows' HWND is a typedef on struct HWND__*
23 #if defined(SFML_SYSTEM_WINDOWS)
24 struct HWND__;
```
namespace sf {

#if defined(SFML_SYSTEM_WINDOWS)
   // Window handle is HWND (HWND__) on Windows
   typedef HWND__* WindowHandle;
#elif defined(SFML_SYSTEM_LINUX) || defined(SFML_SYSTEM_FREEBSD)
   // Window handle is Window (unsigned long) on Unix - X11
   typedef unsigned long WindowHandle;
#elif defined(SFML_SYSTEM_MACOS)
   // Window handle is NSWindow or NSView (void*) on Mac OS X -
   typedef void* WindowHandle;
#elif defined(SFML_SYSTEM_IOS)
   // Window handle is UIWindow (void*) on iOS - UIKit
   typedef void* WindowHandle;
#elif defined(SFML_SYSTEM_ANDROID)
   // Window handle is ANativeWindow* (void*) on Android
   typedef void* WindowHandle;
#elif defined(SFML_DOXYGEN)
   // Define typedef symbol so that Doxygen can attach some doc
   typedef "platform-specific" WindowHandle;
#endif

} // namespace sf

#endif // SFML_WINDOWHANDLE_HPP
/* SFML - Simple and Fast Multimedia Library
   * Copyright (C) 2007-2017 Laurent Gomila (laurent@sfml-dev.org)
   */

#include "SFML_WindowStyle.hpp"
namespace sf {
  namespace Style {
    enum
    {
    ...
    }
}
40  { None = 0,
41    Titlebar = 1 << 0,
42    Resize = 1 << 1,
43    Close = 1 << 2,
44    Fullscreen = 1 << 3,
45    Default = Titlebar | Resize | Close
46  }
47 
48 }
49  // namespace sf
50 
51
52 #endif // SFML_WINDOWSTYLE_HPP
Related Pages

Here is a list of all related documentation pages:

- Deprecated List

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::AlResource Member List

This is the complete list of members for sf::AlResource, including all inherited members.

- AlResource() sf::AlResource [protected]
- ~AlResource() sf::AlResource [protected]

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::InputSoundFile Member List

This is the complete list of members for sf::InputSoundFile, including all in

- `getChannelCount()` const
- `getDuration()` const
- `getSampleCount()` const
- `getSampleRate()` const
- `InputSoundFile()`
- `NonCopyable()`
- `openForWriting(const std::string &filename, unsigned int channelCount, unsigned int sampleRate)`
- `openFromFile(const std::string &filename)`
- `openFromMemory(const void *data, std::size_t sizeInBytes)`
- `openFromStream(InputStream &stream)`
- `read(Int16 *samples, Uint64 maxCount)`
- `seek(Uint64 sampleOffset)`
- `seek(Time timeOffset)`
- `~InputSoundFile()`
sf::Listener Member List

This is the complete list of members for sf::Listener, including all inherited

- `getDirection()`
- `getGlobalVolume()`
- `getPosition()`
- `getUpVector()`
- `setDirection(float x, float y, float z)`
- `setDirection(const Vector3f &direction)`
- `setGlobalVolume(float volume)`
- `setPosition(float x, float y, float z)`
- `setPosition(const Vector3f &position)`
- `setUpVector(float x, float y, float z)`
- `setUpVector(const Vector3f &upVector)`
This is the complete list of members for `sf::Music`, including all inherited members:

- `getAttenuation() const`
- `getChannelCount() const`
- `getDuration() const`
- `getLoop() const`
- `getMinDistance() const`
- `getPitch() const`
- `getPlayingOffset() const`
- `getPosition() const`
- `getSampleRate() const`
- `getStatus() const`
- `getVolume() const`
- `sf::SoundStream::initialize(unsigned int channelCount, unsigned int sampleRate)`
- `isRelativeToListener() const`
- `m_source`
Music()

onGetData(Chunk &data)
onSeek(Time timeOffset)
openFromFile(const std::string &filename)
openFromMemory(const void *data, std::size_t sizeInBytes)
openFromStream(InputStream &stream)
operator=(const SoundSource &right)
pause()

Paused enum value

play()

Playing enum value

setAttenuation(float attenuation)
setLoop(bool loop)
setMinDistance(float distance)
setPitch(float pitch)
setPlayingOffset(Time timeOffset)
setPosition(float x, float y, float z)
setPosition(const Vector3f &position)
setRelativeToListener(bool relative)
setVolume(float volume)

SoundSource(const SoundSource &copy)

SoundSource()

SoundStream()

Status enum name

stop()

Stopped enum value
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~Music()</code></td>
</tr>
<tr>
<td><code>~SoundSource()</code></td>
</tr>
<tr>
<td><code>~SoundStream()</code></td>
</tr>
</tbody>
</table>
sf::OutputSoundFile Member List

This is the complete list of members for `sf::OutputSoundFile`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonCopyable()</td>
</tr>
<tr>
<td>openFromFile(const std::string &amp;filename, unsigned int sampleRate, unsigned int channelCount)</td>
</tr>
<tr>
<td>OutputSoundFile()</td>
</tr>
<tr>
<td>write(const Int16 *samples, Uint64 count)</td>
</tr>
<tr>
<td>~OutputSoundFile()</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Sound Member List

This is the complete list of members for sf::Sound, including all inherited members.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAttenuation() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getBuffer() const</td>
<td>sf::Sound</td>
</tr>
<tr>
<td>getLoop() const</td>
<td>sf::Sound</td>
</tr>
<tr>
<td>getMinDistance() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getPitch() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getPlayingOffset() const</td>
<td>sf::Sound</td>
</tr>
<tr>
<td>getPosition() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getStatus() const</td>
<td>sf::Sound</td>
</tr>
<tr>
<td>getVolume() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>isRelativeToListener() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>m_source</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>operator=(const Sound &amp;right)</td>
<td>sf::Sound</td>
</tr>
<tr>
<td>sf::SoundSource::operator=(const SoundSource &amp;right)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>pause()</td>
<td>sf::Sound</td>
</tr>
<tr>
<td>Method</td>
<td>Class</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td><strong>Paused</strong> enum value** play()**</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>Playing</strong> enum value** resetBuffer()**</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>setAttenuation(float attenuation)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>setBuffer(const SoundBuffer &amp;buffer)</strong></td>
<td>sf::Sound</td>
</tr>
<tr>
<td><strong>setLoop(bool loop)</strong></td>
<td>sf::Sound</td>
</tr>
<tr>
<td><strong>setMinDistance(float distance)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>setPitch(float pitch)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>setPlayingOffset(Time timeOffset)</strong></td>
<td>sf::Sound</td>
</tr>
<tr>
<td><strong>setPosition(float x, float y, float z)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>setPosition(const Vector3f &amp;position)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>setRelativeToListener(bool relative)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>setVolume(float volume)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>Sound()</strong></td>
<td>sf::Sound</td>
</tr>
<tr>
<td><strong>Sound(const SoundBuffer &amp;buffer)</strong></td>
<td>sf::Sound</td>
</tr>
<tr>
<td><strong>Sound(const Sound &amp;copy)</strong></td>
<td>sf::Sound</td>
</tr>
<tr>
<td><strong>SoundSource(const SoundSource &amp;copy)</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>SoundSource()</strong></td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>Status</strong> enum name** stop()**</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>Stopped</strong> enum value** ~Sound()**</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td><strong>~SoundSource()</strong></td>
<td>sf::SoundSource</td>
</tr>
</tbody>
</table>
sf::SoundBuffer Member List

This is the complete list of members for sf::SoundBuffer, including all inhe

AIResource()
getChannelCount() const
getDuration() const
getSampleCount() const
getSampleRate() const
getSamples() const
loadFromFile(const std::string &filename)
loadFromMemory(const void *data, std::size_t sizeInBytes)
loadFromSamples(const Int16 *samples, Uint64 sampleCount, unsigned int channelCount, unsigned int sampleRate)
loadFromStream(InputStream &stream)
operator=(const SoundBuffer &right)
saveToFile(const std::string &filename) const

Sound (defined in sf::SoundBuffer)

SoundBuffer()
SoundBuffer(const SoundBuffer &copy)
~AIResource()
~SoundBuffer()
### sf::SoundBufferRecorder Member List

This is the complete list of members for `sf::SoundBufferRecorder`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getAvailableDevices()</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>getBuffer()</code> const</td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>getChannelCount()</code> const</td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>getDefaultDevice()</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>getDevice()</code> const</td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>getSampleRate()</code> const</td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>isAvailable()</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>onProcessSamples(const Int16 *samples, std::size_t sampleCount)</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>onStart()</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>onStop()</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>setChannelCount(unsigned int channelCount)</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>setDevice(const std::string &amp;name)</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>setProcessingInterval(Time interval)</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
<tr>
<td><code>SoundRecorder()</code></td>
<td>sf::SoundBufferRecorder</td>
</tr>
</tbody>
</table>
start(unsigned int sampleRate=44100)

stop()

~SoundBufferRecorder()

~SoundRecorder()
### sf::SoundFileFactory Member List

This is the complete list of members for `sf::SoundFileFactory`, including all inherited members:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createReaderFromFilename(const std::string &amp;filename)</code></td>
<td>sf::SoundFileFactory</td>
</tr>
<tr>
<td><code>createReaderFromMemory(const void *data, std::size_t sizeInBytes)</code></td>
<td>sf::SoundFileFactory</td>
</tr>
<tr>
<td><code>createReaderFromStream(InputStream &amp;stream)</code></td>
<td>sf::SoundFileFactory</td>
</tr>
<tr>
<td><code>createWriterFromFilename(const std::string &amp;filename)</code></td>
<td>sf::SoundFileFactory</td>
</tr>
<tr>
<td><code>registerReader()</code></td>
<td>sf::SoundFileFactory</td>
</tr>
<tr>
<td><code>registerWriter()</code></td>
<td>sf::SoundFileFactory</td>
</tr>
<tr>
<td><code>unregisterReader()</code></td>
<td>sf::SoundFileFactory</td>
</tr>
<tr>
<td><code>unregisterWriter()</code></td>
<td>sf::SoundFileFactory</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::SoundFileReader Member List

This is the complete list of members for sf::SoundFileReader, including all

- `open(InputStream &stream, Info &info)=0`
- `read(Int16 *samples, Uint64 maxCount)=0`
- `seek(Uint64 sampleOffset)=0`
- `~SoundFileReader()`
sf::SoundFileReader::Info Member List

This is the complete list of members for sf::SoundFileReader::Info, including all inherited members.

- channelCount
- sampleCount
- sampleRate

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::SoundFileWriter Member List

This is the complete list of members for sf::SoundFileWriter, including all inherited members.

- `open(const std::string &filename, unsigned int sampleRate, unsigned int channelCount)=0`
- `write(const Int16 *samples, Uint64 count)=0`
- `~SoundFileWriter()`

SFML is licensed under the terms and conditions of the zlib/png license. Copyright © Laurent Gomila. :: Documentation generated by doxygen.
sf::SoundRecorder Member List

This is the complete list of members for sf::SoundRecorder, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlResource()</td>
<td>sf::AlResource()</td>
</tr>
<tr>
<td>getAvailableDevices()</td>
<td>sf::SoundRecorder::getAvailableDevices()</td>
</tr>
<tr>
<td>getChannelCount() const</td>
<td>sf::SoundRecorder::getChannelCount() const</td>
</tr>
<tr>
<td>getDefaultDevice()</td>
<td>sf::SoundRecorder::getDefaultDevice()</td>
</tr>
<tr>
<td>getDevice() const</td>
<td>sf::SoundRecorder::getDevice() const</td>
</tr>
<tr>
<td>getSampleRate() const</td>
<td>sf::SoundRecorder::getSampleRate() const</td>
</tr>
<tr>
<td>isAvailable()</td>
<td>sf::SoundRecorder::isAvailable()</td>
</tr>
<tr>
<td>onProcessSamples(const Int16 *samples, std::size_t sampleCount)=0</td>
<td>sf::SoundRecorder::onProcessSamples()</td>
</tr>
<tr>
<td>onStart()</td>
<td>sf::SoundRecorder::onStart()</td>
</tr>
<tr>
<td>onStop()</td>
<td>sf::SoundRecorder::onStop()</td>
</tr>
<tr>
<td>setChannelCount(unsigned int channelCount)</td>
<td>sf::SoundRecorder::setChannelCount(unsigned int channelCount)</td>
</tr>
<tr>
<td>setDevice(const std::string &amp;name)</td>
<td>sf::SoundRecorder::setDevice(const std::string &amp;name)</td>
</tr>
<tr>
<td>setProcessingInterval(Time interval)</td>
<td>sf::SoundRecorder::setProcessingInterval(Time interval)</td>
</tr>
<tr>
<td>SoundRecorder()</td>
<td>sf::SoundRecorder()</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td><code>start(unsigned int sampleRate=44100)</code></td>
<td>sf:</td>
</tr>
<tr>
<td><code>stop()</code></td>
<td>sf:</td>
</tr>
<tr>
<td><code>~AlResource()</code></td>
<td>sf:</td>
</tr>
<tr>
<td><code>~SoundRecorder()</code></td>
<td>sf:</td>
</tr>
</tbody>
</table>
This is the complete list of members for sf::SoundSource, including all inherited members:

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlResource()</td>
<td>sf::AlResource</td>
</tr>
<tr>
<td>getAttenuation() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getMinDistance() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getPitch() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getPosition() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getStatus() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>getVolume() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>isRelativeToListener() const</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>m_source</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>operator=(const SoundSource &amp;right)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Paused enum value</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Playing enum value</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setAttenuation(float attenuation)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setMinDistance(float distance)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Method</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>setPitch(float pitch)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setPosition(float x, float y, float z)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setPosition(const Vector3f &amp;position)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setRelativeToListener(bool relative)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setVolume(float volume)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>SoundSource(const SoundSource &amp;copy)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>SoundSource()</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Status enum name</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Stopped enum value</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>~AIResource()</td>
<td>sf::AIResource</td>
</tr>
<tr>
<td>~SoundSource()</td>
<td>sf::SoundSource</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::SoundStream Member List

This is the complete list of members for sf::SoundStream, including all inherited members:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAttenuation() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getChannelCount() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getLoop() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getMinDistance() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getPitch() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getPlayingOffset() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getPosition() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getSampleRate() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getStatus() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>getVolume() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>initialize(unsigned int channelCount, unsigned int sampleRate)</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>isRelativeToListener() const</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>m_source</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>onGetData(Chunk &amp;data)=0</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>onSeek(Time timeOffset)=0</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>operator=(const SoundSource &amp;right)</td>
<td>sf::SoundStream</td>
</tr>
<tr>
<td>pause()</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Paused</td>
<td>enum value</td>
</tr>
<tr>
<td>play()</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Playing</td>
<td>enum value</td>
</tr>
<tr>
<td>setAttenuation(float attenuation)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setLoop(bool loop)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setMinDistance(float distance)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setPitch(float pitch)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setPlayingOffset(Time timeOffset)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setPosition(float x, float y, float z)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setPosition(const Vector3f &amp;position)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setRelativeToListener(bool relative)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>setVolume(float volume)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>SoundSource(const SoundSource &amp;copy)</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>SoundSource()</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>SoundStream()</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Status</td>
<td>enum name</td>
</tr>
<tr>
<td>stop()</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>Stopped</td>
<td>enum value</td>
</tr>
<tr>
<td>~SoundSource()</td>
<td>sf::SoundSource</td>
</tr>
<tr>
<td>~SoundStream()</td>
<td>sf::SoundSource</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
sf::SoundStream::Chunk Member List

This is the complete list of members for sf::SoundStream::Chunk, including:

- `sampleCount` sf::SoundStream::Chunk
- `samples` sf::SoundStream::Chunk

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::BlendMode Member List

This is the complete list of members for sf::BlendMode, including all inherited members:

- Add enum value
- alphaDstFactor
- alphaEquation
- alphaSrcFactor
- BlendMode()
- BlendMode(Factor sourceFactor, Factor destinationFactor, Equation blendEquation)
- BlendMode(Factor colorSourceFactor, Factor colorDestinationFactor, Equation colorBlendEquation, Factor alphaSourceFactor, Factor alphaDestinationFactor, Equation alphaBlendEquation)
- colorDstFactor
- colorEquation
- colorSrcFactor
- DstAlpha enum value
- DstColor enum value
- Equation enum name
- Factor enum name
One enum value
OneMinusDstAlpha enum value
OneMinusDstColor enum value
OneMinusSrcAlpha enum value
OneMinusSrcColor enum value
operator!=(const BlendMode &left, const BlendMode &right)
operator==(const BlendMode &left, const BlendMode &right)
ReverseSubtract enum value
SrcAlpha enum value
SrcColor enum value
Subtract enum value
Zero enum value
sf::CircleShape Member List

This is the complete list of members for sf::CircleShape, including all inherited members:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Class Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CircleShape(float radius=0, std::size_t pointCount=30)</td>
<td>sf::CircleShape</td>
</tr>
<tr>
<td>getFillColor() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getGlobalBounds() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getInverseTransform() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getLocalBounds() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getOrigin() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getOutlineColor() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getOutlineThickness() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getPoint(std::size_t index) const</td>
<td>sf::CircleShape</td>
</tr>
<tr>
<td>getPointCount() const</td>
<td>sf::CircleShape</td>
</tr>
<tr>
<td>getPosition() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getRadius() const</td>
<td>sf::CircleShape</td>
</tr>
<tr>
<td>getRotation() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getScale() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>~Shape()</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>~Transformable()</td>
<td>sf::Transformable</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
This is the complete list of members for `sf::Color`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td><code>sf::Color</code></td>
</tr>
<tr>
<td>b</td>
<td><code>sf::Color</code></td>
</tr>
<tr>
<td>Black</td>
<td><code>sf::Color</code> static</td>
</tr>
<tr>
<td>Blue</td>
<td><code>sf::Color</code> static</td>
</tr>
<tr>
<td>Color()</td>
<td><code>sf::Color</code></td>
</tr>
<tr>
<td>Color(Uint8 red, Uint8 green, Uint8 blue, Uint8 alpha=255)</td>
<td><code>sf::Color</code> explicit</td>
</tr>
<tr>
<td>Color(Uint32 color)</td>
<td><code>sf::Color</code> static</td>
</tr>
<tr>
<td>Cyan</td>
<td><code>sf::Color</code> static</td>
</tr>
<tr>
<td>g</td>
<td><code>sf::Color</code> static</td>
</tr>
<tr>
<td>Green</td>
<td><code>sf::Color</code> static</td>
</tr>
<tr>
<td>Magenta</td>
<td><code>sf::Color</code> static</td>
</tr>
<tr>
<td>operator!=(const Color &amp;left, const Color &amp;right)</td>
<td><code>sf::Color</code> related</td>
</tr>
<tr>
<td>operator*(const Color &amp;left, const Color &amp;right)</td>
<td><code>sf::Color</code> related</td>
</tr>
<tr>
<td>operator*=(Color &amp;left, const Color &amp;right)</td>
<td><code>sf::Color</code> related</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>operator+</td>
<td>(const Color &amp;left, const Color &amp;right)</td>
</tr>
<tr>
<td>operator+=</td>
<td>(Color &amp;left, const Color &amp;right)</td>
</tr>
<tr>
<td>operator-</td>
<td>(const Color &amp;left, const Color &amp;right)</td>
</tr>
<tr>
<td>operator-=</td>
<td>(Color &amp;left, const Color &amp;right)</td>
</tr>
<tr>
<td>operator==</td>
<td>(const Color &amp;left, const Color &amp;right)</td>
</tr>
</tbody>
</table>

r

<table>
<thead>
<tr>
<th>Color</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>sf::Color static</td>
</tr>
<tr>
<td>toInteger() const</td>
<td>sf::Color static</td>
</tr>
<tr>
<td>Transparent</td>
<td>sf::Color static</td>
</tr>
<tr>
<td>White</td>
<td>sf::Color static</td>
</tr>
<tr>
<td>Yellow</td>
<td>sf::Color static</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
### sf::ConvexShape Member List

This is the complete list of members for `sf::ConvexShape`, including all inherited members.

<table>
<thead>
<tr>
<th>Member Description</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConvexShape(std::size_t pointCount=0)</code></td>
<td><code>sf::ConvexShape</code></td>
</tr>
<tr>
<td><code>getFillColor() const</code></td>
<td><code>sf::Shape</code></td>
</tr>
<tr>
<td><code>getGlobalBounds() const</code></td>
<td><code>sf::Shape</code></td>
</tr>
<tr>
<td><code>getInverseTransform() const</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getLocalBounds() const</code></td>
<td><code>sf::Shape</code></td>
</tr>
<tr>
<td><code>getOrigin() const</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getOutlineColor() const</code></td>
<td><code>sf::Shape</code></td>
</tr>
<tr>
<td><code>getOutlineThickness() const</code></td>
<td><code>sf::Shape</code></td>
</tr>
<tr>
<td><code>getPoint(std::size_t index) const</code></td>
<td><code>sf::ConvexShape</code></td>
</tr>
<tr>
<td><code>getPointCount() const</code></td>
<td><code>sf::ConvexShape</code></td>
</tr>
<tr>
<td><code>getPosition() const</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getRotation() const</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getScale() const</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getTexture() const</code></td>
<td><code>sf::Shape</code></td>
</tr>
</tbody>
</table>
### sf::Transformable

- `getTextureRect() const` sf::Shape
- `getTransform() const` sf::Transformable
- `move(float offsetX, float offsetY)` sf::Transformable
- `move(const Vector2f &offset)` sf::Transformable
- `rotate(float angle)` sf::Transformable
- `scale(float factorX, float factorY)` sf::Transformable
- `scale(const Vector2f &factor)` sf::Transformable
- `setFillColor(const Color &color)` sf::Shape
- `setOrigin(float x, float y)` sf::Transformable
- `setOrigin(const Vector2f &origin)` sf::Transformable
- `setOutlineColor(const Color &color)` sf::Shape
- `setOutlineThickness(float thickness)` sf::Shape
- `setPoint(std::size_t index, const Vector2f &point)` sf::ConvexShape
- `setPointCount(std::size_t count)` sf::ConvexShape
- `setPosition(float x, float y)` sf::Transformable
- `setPosition(const Vector2f &position)` sf::Transformable
- `setRotation(float angle)` sf::Transformable
- `setScale(float factorX, float factorY)` sf::Transformable
- `setScale(const Vector2f &factors)` sf::Transformable
- `setTexture(const Texture *texture, bool resetRect=false)` sf::Shape
- `setTextureRect(const IntRect &rect)` sf::Shape
- `Shape()` sf::Shape
- `Transformable()` sf::Transformable
- `update()` sf::Shape
- `~Drawable()` sf::Drawable
- `~Shape()` sf::Shape
~Transformable()
sf::Drawable Member List

This is the complete list of members for sf::Drawable, including all inherit:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>draw(RenderTarget &amp;target, RenderStates states) const =0</code></td>
<td>sf::Drawable</td>
</tr>
<tr>
<td><code>RenderTarget (defined in sf::Drawable)</code></td>
<td>sf::Drawable</td>
</tr>
<tr>
<td><code>~Drawable()</code></td>
<td>sf::Drawable</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Font Member List

This is the complete list of members for sf::Font, including all inherited members.

Font()
Font(const Font &copy)
getGlyph(Uint32 codePoint, unsigned int characterSize, bool bold, float outlineThickness=0) const
getInfo() const
getKerning(Uint32 first, Uint32 second, unsigned int characterSize) const
getLineSpacing(unsigned int characterSize) const
getTexture(unsigned int characterSize) const
getUnderlinePosition(unsigned int characterSize) const
getUnderlineThickness(unsigned int characterSize) const
loadFromFile(const std::string &filename)
loadFromMemory(const void *data, std::size_t sizeInBytes)
loadFromStream(InputStream &stream)
operator=(const Font &right)
~Font()
sf::Font::Info Member List

This is the complete list of members for sf::Font::Info, including all inherited members.

family sf::Font::Info
sf::Glyph Member List

This is the complete list of members for sf::Glyph, including all inherited members.

<table>
<thead>
<tr>
<th>advance</th>
<th>sf::Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>bounds</td>
<td>sf::Glyph</td>
</tr>
<tr>
<td>Glyph()</td>
<td>sf::Glyph</td>
</tr>
<tr>
<td>textureRect</td>
<td>sf::Glyph</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Image Member List

This is the complete list of members for sf::Image, including all inherited members:

- `copy(const Image &source, unsigned int destX, unsigned int destY, const IntRect &sourceRect=IntRect(0, 0, 0, 0), bool applyAlpha=false)`
- `create(unsigned int width, unsigned int height, const Color &color=Color(0, 0, 0))`
- `create(unsigned int width, unsigned int height, const Uint8 *pixels)`
- `createMaskFromColor(const Color &color, Uint8 alpha=0)`
- `flipHorizontally()`
- `flipVertically()`
- `getPixel(unsigned int x, unsigned int y) const`
- `getPixelsPtr() const`
- `getSize() const`
- `Image()`
- `loadFromFile(const std::string &filename)`
- `loadFromMemory(const void *data, std::size_t size)`
- `loadFromStream(InputStream &stream)`
- `saveToFile(const std::string &filename) const`
setPixel(unsigned int x, unsigned int y, const Color &color)

~Image()
sf::Rect< T > Member List

This is the complete list of members for `sf::Rect< T >`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contains(T x, T y) const</code></td>
<td>sf::Rect&lt; T &gt; contains(T x, T y) const</td>
</tr>
<tr>
<td><code>contains(const Vector2&lt; T &gt;&amp;point) const</code></td>
<td>sf::Rect&lt; T &gt; contains(const Vector2&lt; T &gt;&amp;point) const</td>
</tr>
<tr>
<td><code>height</code></td>
<td>sf::Rect&lt; T &gt; height</td>
</tr>
<tr>
<td><code>intersects(const Rect&lt; T &gt;&amp;rectangle) const</code></td>
<td>sf::Rect&lt; T &gt; intersects(const Rect&lt; T &gt;&amp;rectangle) const</td>
</tr>
<tr>
<td><code>intersects(const Rect&lt; T &gt;&amp;rectangle, Rect&lt; T &gt;&amp;intersection) const</code></td>
<td>sf::Rect&lt; T &gt; intersects(const Rect&lt; T &gt;&amp;rectangle, Rect&lt; T &gt;&amp;intersection) const</td>
</tr>
<tr>
<td><code>left</code></td>
<td>sf::Rect&lt; T &gt; left</td>
</tr>
<tr>
<td><code>operator!=(const Rect&lt; T &gt;&amp;left, const Rect&lt; T &gt;&amp;right)</code></td>
<td>sf::Rect&lt; T &gt; operator!=(const Rect&lt; T &gt;&amp;left, const Rect&lt; T &gt;&amp;right)</td>
</tr>
<tr>
<td><code>operator==(const Rect&lt; T &gt;&amp;left, const Rect&lt; T &gt;&amp;right)</code></td>
<td>sf::Rect&lt; T &gt; operator==(const Rect&lt; T &gt;&amp;left, const Rect&lt; T &gt;&amp;right)</td>
</tr>
<tr>
<td><code>Rect()</code></td>
<td>sf::Rect&lt; T &gt; Rect()</td>
</tr>
<tr>
<td><code>Rect(T rectLeft, T rectTop, T rectWidth, T rectHeight)</code></td>
<td>sf::Rect&lt; T &gt; Rect(T rectLeft, T rectTop, T rectWidth, T rectHeight)</td>
</tr>
<tr>
<td><code>Rect(const Vector2&lt; T &gt;&amp;position, const Vector2&lt; T &gt;&amp;size)</code></td>
<td>sf::Rect&lt; T &gt; Rect(const Vector2&lt; T &gt;&amp;position, const Vector2&lt; T &gt;&amp;size)</td>
</tr>
<tr>
<td><code>Rect(const Rect&lt; U &gt;&amp;rectangle)</code></td>
<td>sf::Rect&lt; T &gt; Rect(const Rect&lt; U &gt;&amp;rectangle)</td>
</tr>
<tr>
<td><code>top</code></td>
<td>sf::Rect&lt; T &gt; top</td>
</tr>
<tr>
<td><code>width</code></td>
<td>sf::Rect&lt; T &gt; width</td>
</tr>
</tbody>
</table>
sf::RectangleShape Member List

This is the complete list of members for `sf::RectangleShape`, including all

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>getFillColor() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getGlobalBounds() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getInverseTransform() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getLocalBounds() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getOrigin() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getOutlineColor() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getOutlineThickness() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getPoint(std::size_t index) const</td>
<td>sf::RectangleShape</td>
</tr>
<tr>
<td>getPointCount() const</td>
<td>sf::RectangleShape</td>
</tr>
<tr>
<td>getPosition() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getRotation() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getScale() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getSize() const</td>
<td>sf::RectangleShape</td>
</tr>
<tr>
<td>getTexture() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>Function</td>
<td>Class</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>getTextureRect() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getTransform() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>move(float offsetX, float offsetY)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>move(const Vector2f &amp;offset)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>RectangleShape(const Vector2f &amp;size=Vector2f(0, 0))</td>
<td>sf::RectangleShape</td>
</tr>
<tr>
<td>rotate(float angle)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>scale(float factorX, float factorY)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>scale(const Vector2f &amp;factor)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setFillColor(const Color &amp;color)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setOrigin(float x, float y)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setOrigin(const Vector2f &amp;origin)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setOutlineColor(const Color &amp;color)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setOutlineThickness(float thickness)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setPosition(float x, float y)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setPosition(const Vector2f &amp;position)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setRotation(float angle)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setScale(float factorX, float factorY)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setScale(const Vector2f &amp;factors)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setSize(const Vector2f &amp;size)</td>
<td>sf::RectangleShape</td>
</tr>
<tr>
<td>setTexture(const Texture *texture, bool resetRect=false)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setTextureRect(const IntRect &amp;rect)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>Shape()</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>Transformable()</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>update()</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>~Drawable()</td>
<td>sf::Drawable</td>
</tr>
<tr>
<td>~Shape()</td>
<td>sf::Shape</td>
</tr>
</tbody>
</table>
~Transformable() sf::Transformable

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::RenderStates Member List

This is the complete list of members for `sf::RenderStates`, including all inherited members:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>blendMode</code></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td><code>RenderStates()</code></td>
<td></td>
</tr>
<tr>
<td><code>const BlendMode &amp;theBlendMode</code></td>
<td></td>
</tr>
<tr>
<td><code>const Transform &amp;theTransform</code></td>
<td></td>
</tr>
<tr>
<td><code>const Texture *theTexture</code></td>
<td></td>
</tr>
<tr>
<td><code>const Shader *theShader</code></td>
<td></td>
</tr>
<tr>
<td><code>RenderStates(const BlendMode &amp;theBlendMode, const Transform &amp;theTransform)</code></td>
<td></td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila. :: Documentation generated by doxygen.
sf::RenderTarget Member List

This is the complete list of members for sf::RenderTarget, including all inherited members.

- `clear(const Color &color=Color(0, 0, 0, 255))`
- `draw(const Drawable &drawable, const RenderStates &states=RenderStates::Default)`
- `draw(const Vertex *vertices, std::size_t vertexCount, PrimitiveType type, const RenderStates &states=RenderStates::Default)`
- `getDefaultView() const`
- `getSize() const =0`
- `getView() const`
- `getViewport(const View &view) const`
- `initialize()`
- `mapCoordsToPixel(const Vector2f &point) const`
- `mapCoordsToPixel(const Vector2f &point, const View &view) const`
- `mapPixelToCoords(const Vector2i &point) const`
- `mapPixelToCoords(const Vector2i &point, const View &view) const`
- `NonCopyable()`
- `popGLStates()`
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pushGLStates()</code></td>
<td></td>
</tr>
<tr>
<td><code>RenderTarget()</code></td>
<td></td>
</tr>
<tr>
<td><code>resetGLStates()</code></td>
<td></td>
</tr>
<tr>
<td><code>getView()</code></td>
<td>const View &amp;view</td>
</tr>
<tr>
<td><code>~RenderTarget()</code></td>
<td></td>
</tr>
</tbody>
</table>
sf::RenderTexture Member List

This is the complete list of members for sf::RenderTexture, including all inherited members.

- clear(const Color &color=Color(0, 0, 0, 255))
- create(unsigned int width, unsigned int height, bool depthBuffer=false)
- display()
- draw(const Drawable &drawable, const RenderStates &states=RenderStates::Default)
- draw(const Vertex *vertices, std::size_t vertexCount, PrimitiveType type, const RenderStates &states=RenderStates::Default)
- generateMipmap()
- getDefaultView() const
- getSize() const
- getTexture() const
- getView() const
- getViewport(const View &view) const
- initialize()
- isRepeated() const
- isSmooth() const
mapCoordsToPixel(const Vector2f &point) const
mapCoordsToPixel(const Vector2f &point, const View &view) const
mapPixelToCoords(const Vector2i &point) const
mapPixelToCoords(const Vector2i &point, const View &view) const
popGLStates()
pushGLStates()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
RenderTarget()
sf::RenderWindow Member List

This is the complete list of members for sf::RenderWindow, including all inherited members.

capture() const
clear(const Color &color=Color(0, 0, 0, 255))
close()
create (VideoMode mode, const String &title, Uint32 style=Style::Default, const ContextSettings &settings)
create(WindowHandle handle, const ContextSettings &settings)
display()
draw(const Drawable &drawable, const RenderStates &states=RenderStates::Default)
draw(const Vertex *, std::size_t vertexCount, PrimitiveType type, const RenderStates &states)
defaultView() const
gPosition() const
gSettings() const
gSize() const
gSystemHandle() const
gView() const
getViewport(const View &view) const

hasFocus() const

initialize()

isOpen() const

mapCoordsToPixel(const Vector2f &point) const

mapCoordsToPixel(const Vector2f &point, const View &view) const

mapPixelToCoords(const Vector2i &point) const

mapPixelToCoords(const Vector2i &point, const View &view) const

NonCopyable()

onCreate()

onResize()

pollEvent(Event &event)

popGLStates()

pushGLStates()

RenderTarget()

RenderWindow()

RenderWindow(VideoMode mode, const String &title, Uint32 style=Style::Default, const ContextSettings &settings=ContextSettings())

RenderWindow(WindowHandle handle, const ContextSettings &settings=ContextSettings())

requestFocus()

resetGLStates()

setActive(bool active=true) const

setFramerateLimit(unsigned int limit)

setIcon(unsigned int width, unsigned int height, const Uint8 *pixels)

setJoystickThreshold(float threshold)

setKeyRepeatEnabled(bool enabled)

setMouseCursorGrabbed(bool grabbed)
setMouseCursorVisible(bool visible)
setPosition(const Vector2i &position)
setSize(const Vector2u &size)
setTitle(const String &title)
setVerticalSyncEnabled(bool enabled)
setView(const View &view)
setVisible(bool visible)
waitEvent(Event &event)

Window()
Window(VideoMode mode, const String &title, Uint32 style=Style::Default)
Window(WindowHandle handle, const ContextSettings &settings=ContextSettings())
~RenderTarget()
~RenderWindow()
~Window()
sf::Shader Member List

This is the complete list of members for sf::Shader, including all inherited members:

- `bind(const Shader *shader)`
- `CurrentTexture`
- `ensureGlContext()`
- `Fragment enum value`
- `Geometry enum value`
- `getNativeHandle() const`
- `GlResource()`
- `isAvailable()`
- `isGeometryAvailable()`
- `loadFromFile(const std::string &filename, Type type)`
- `loadFromFile(const std::string &vertexShaderFilename, const std::string &fragmentShaderFilename)`
- `loadFromFile(const std::string &vertexShaderFilename, const std::string &geometryShaderFilename, const std::string &fragmentShaderFilename)`
- `loadFromMemory(const std::string &shader, Type type)`
- `loadFromMemory(const std::string &vertexShader, const std::string &fragmentShader)`
loadFromMemory(const std::string &vertexShader, const std::string &geometryShader, const std::string &fragmentShader)

loadFromStream(InputStream &stream, Type type)

loadFromStream(InputStream &vertexShaderStream, InputStream &fragmentShaderStream)

loadFromStream(InputStream &vertexShaderStream, InputStream &geometryShaderStream, InputStream &fragmentShaderStream)

NonCopyable()

setParameter(const std::string &name, float x)

setParameter(const std::string &name, float x, float y)

setParameter(const std::string &name, float x, float y, float z)

setParameter(const std::string &name, float x, float y, float z, float w)

setParameter(const std::string &name, const Vector2f &vector)

setParameter(const std::string &name, const Vector3f &vector)

setParameter(const std::string &name, const Color &color)

setParameter(const std::string &name, const Transform &transform)

setParameter(const std::string &name, const Texture &texture)

setParameter(const std::string &name, CurrentTextureType)

setUniform(const std::string &name, float x)

setUniform(const std::string &name, const Glsl::Vec2 &vector)

setUniform(const std::string &name, const Glsl::Vec3 &vector)

setUniform(const std::string &name, const Glsl::Vec4 &vector)

setUniform(const std::string &name, int x)

setUniform(const std::string &name, const Glsl::Ivec2 &vector)

setUniform(const std::string &name, const Glsl::Ivec3 &vector)

setUniform(const std::string &name, const Glsl::Ivec4 &vector)

setUniform(const std::string &name, bool x)

setUniform(const std::string &name, const Glsl::Bvec2 &vector)

setUniform(const std::string &name, const Glsl::Bvec3 &vector)
setUniform(const std::string &name, const Glsl::Bvec4 &vector)
setUniform(const std::string &name, const Glsl::Mat3 &matrix)
setUniform(const std::string &name, const Glsl::Mat4 &matrix)
setUniform(const std::string &name, const Texture &texture)
setUniform(const std::string &name, CurrentTextureType)
setUniformArray(const std::string &name, const float *scalarArray, std::size_t length)
setUniformArray(const std::string &name, const Glsl::Vec2 *vectorArray, std::size_t length)
setUniformArray(const std::string &name, const Glsl::Vec3 *vectorArray, std::size_t length)
setUniformArray(const std::string &name, const Glsl::Vec4 *vectorArray, std::size_t length)
setUniformArray(const std::string &name, const Glsl::Mat3 *matrixArray, std::size_t length)
setUniformArray(const std::string &name, const Glsl::Mat4 *matrixArray, std::size_t length)

Shader()

Type enum name

Vertex enum value

~GLResource()

~Shader()
sf::Shape Member List

This is the complete list of members for sf::Shape, including all inherited members.

<table>
<thead>
<tr>
<th>Member Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>getFillColor() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getGlobalBounds() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getInverseTransform() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getLocalBounds() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getOrigin() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getOutlineColor() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getOutlineThickness() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getPoint(std::size_t index) const =0</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getPointCount() const =0</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getPosition() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getRotation() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getScale() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>getTexture() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>getTextureRect() const</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>Method</td>
<td>Class</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>getTransform() const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>move(float offsetX, float offsetY)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>move(const Vector2f &amp;offset)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>rotate(float angle)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>scale(float factorX, float factorY)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>scale(const Vector2f &amp;factor)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setFillColor(const Color &amp;color)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setOrigin(float x, float y)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setOrigin(const Vector2f &amp;origin)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setOutlineColor(const Color &amp;color)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setOutlineThickness(float thickness)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setPosition(float x, float y)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setPosition(const Vector2f &amp;position)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setRotation(float angle)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>.setScale(float factorX, float factorY)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>.setScale(const Vector2f &amp;factors)</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>setTexture(const Texture *texture, bool resetRect=false)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>setTextureRect(const IntRect &amp;rect)</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>Shape()</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>Transformable()</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td>update()</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>~Drawable()</td>
<td>sf::Drawable</td>
</tr>
<tr>
<td>~Shape()</td>
<td>sf::Shape</td>
</tr>
<tr>
<td>~Transformable()</td>
<td>sf::Transformable</td>
</tr>
</tbody>
</table>
## sf::Sprite Member List

This is the complete list of members for `sf::Sprite`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getColor()</code> const</td>
<td><code>sf::Sprite</code></td>
</tr>
<tr>
<td><code>getGlobalBounds()</code> const</td>
<td><code>sf::Sprite</code></td>
</tr>
<tr>
<td><code>getInverseTransform()</code> const</td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getLocalBounds()</code> const</td>
<td><code>sf::Sprite</code></td>
</tr>
<tr>
<td><code>getOrigin()</code> const</td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getPosition()</code> const</td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getRotation()</code> const</td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getScale()</code> const</td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>getTexture()</code> const</td>
<td><code>sf::Sprite</code></td>
</tr>
<tr>
<td><code>getTextureRect()</code> const</td>
<td><code>sf::Sprite</code></td>
</tr>
<tr>
<td><code>getTransform()</code> const</td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>move(float offsetX, float offsetY)</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>move(const Vector2f &amp;offset)</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td><code>rotate(float angle)</code></td>
<td><code>sf::Transformable</code></td>
</tr>
<tr>
<td>Function</td>
<td>Class</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>scale(float factorX, float factorY)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>scale(const Vector2f &amp;factor)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setColor(const Color &amp;color)</code></td>
<td>sf::Sprite</td>
</tr>
<tr>
<td><code>setOrigin(float x, float y)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setOrigin(const Vector2f &amp;origin)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setPosition(float x, float y)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setPosition(const Vector2f &amp;position)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setRotation(float angle)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setScale(float factorX, float factorY)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setScale(const Vector2f &amp;factors)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setTexture(const Texture &amp;texture, bool resetRect=false)</code></td>
<td>sf::Sprite</td>
</tr>
<tr>
<td><code>setTextureRect(const IntRect &amp;rectangle)</code></td>
<td>sf::Sprite</td>
</tr>
<tr>
<td><code>Sprite()</code></td>
<td>sf::Sprite</td>
</tr>
<tr>
<td><code>Sprite(const Texture &amp;texture)</code></td>
<td>sf::Sprite</td>
</tr>
<tr>
<td><code>Sprite(const Texture &amp;texture, const IntRect &amp;rectangle)</code></td>
<td>sf::Sprite</td>
</tr>
<tr>
<td><code>Transformable()</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>~Drawable()</code></td>
<td>sf::Drawable</td>
</tr>
<tr>
<td><code>~Transformable()</code></td>
<td>sf::Transformable</td>
</tr>
</tbody>
</table>
# sf::Text Member List

This is the complete list of members for `sf::Text`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong> enum value</td>
</tr>
<tr>
<td><code>findCharacterPos(std::size_t index) const</code></td>
</tr>
<tr>
<td><code>getCharacterSize() const</code></td>
</tr>
<tr>
<td><code>getColor() const</code></td>
</tr>
<tr>
<td><code>getFillColor() const</code></td>
</tr>
<tr>
<td><code>getFont() const</code></td>
</tr>
<tr>
<td><code>getGlobalBounds() const</code></td>
</tr>
<tr>
<td><code>getInverseTransform() const</code></td>
</tr>
<tr>
<td><code>getLocalBounds() const</code></td>
</tr>
<tr>
<td><code>getOrigin() const</code></td>
</tr>
<tr>
<td><code>getOutlineColor() const</code></td>
</tr>
<tr>
<td><code>getOutlineThickness() const</code></td>
</tr>
<tr>
<td><code>getPosition() const</code></td>
</tr>
<tr>
<td><code>getRotation() const</code></td>
</tr>
</tbody>
</table>
getScale() const
getString() const
getStyle() const
getTransform() const
Italic enum value
move(float offsetX, float offsetY)
move(const Vector2f &offset)
Regular enum value
rotate(float angle)
scale(float factorX, float factorY)
scale(const Vector2f &factor)
setCharacterSize(unsigned int size)
setColor(const Color &color)
setFillColor(const Color &color)
setFont(const Font &font)
setOrigin(float x, float y)
setOrigin(const Vector2f &origin)
setOutlineColor(const Color &color)
setOutlineThickness(float thickness)
setPosition(float x, float y)
setPosition(const Vector2f &position)
setRotation(float angle)
setScale(float factorX, float factorY)
setScale(const Vector2f &factors)
setString(const String &string)
setStyle(Uint32 style)
**StrikeThrough** enum value

**Style** enum name

`Text()`

`Text(const String &string, const Font &font, unsigned int characterSize=30)`

**Transformable()**

**Underlined** enum value

~**Drawable()**

~**Transformable()**

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Texture Member List

This is the complete list of members for sf::Texture, including all inherited

\texttt{bind(const Texture *texture, CoordinateType coordinateType=Normalized, CoordinateType enum name)}

\texttt{copyToImage() const}

\texttt{create(unsigned int width, unsigned int height)}

\texttt{ensureGlContext()}

\texttt{generateMipmap()}

\texttt{getMaximumSize()}

\texttt{getNativeHandle() const}

\texttt{getSize() const}

\texttt{GlResource()}

\texttt{isRepeated() const}

\texttt{isSmooth() const}

\texttt{isSrgb() const}

\texttt{loadFromFile(const std::string &filename, const IntRect &area=IntRect())}
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadFromImage</td>
<td>Loads an image into the texture</td>
</tr>
<tr>
<td>loadFromMemory</td>
<td>Loads memory data into the texture</td>
</tr>
<tr>
<td>loadFromStream</td>
<td>Loads data from a stream into the texture</td>
</tr>
<tr>
<td>Normalized enum value</td>
<td>Represent a normalized value</td>
</tr>
<tr>
<td>operator=(const Texture &amp;right)</td>
<td>Assigns a texture to an object</td>
</tr>
<tr>
<td>Pixels enum value</td>
<td>Represent a pixel value</td>
</tr>
<tr>
<td>RenderTarget (defined in sf::Texture)</td>
<td>Represents a texture target</td>
</tr>
<tr>
<td>RenderTexture (defined in sf::Texture)</td>
<td>Represents a texture</td>
</tr>
<tr>
<td>setRepeated(bool repeated)</td>
<td>Sets the texture to be repeated</td>
</tr>
<tr>
<td>setSmooth(bool smooth)</td>
<td>Sets the texture to be smooth</td>
</tr>
<tr>
<td>setSrgb(bool sRgba)</td>
<td>Sets the texture to be sRGB</td>
</tr>
<tr>
<td>Texture()</td>
<td>Default constructor</td>
</tr>
<tr>
<td>Texture(const Texture &amp;copy)</td>
<td>Copy constructor</td>
</tr>
<tr>
<td>update(const Uint8 *pixels)</td>
<td>Updates the texture with raw pixel data</td>
</tr>
<tr>
<td>update(const Uint8 *pixels, unsigned int width, unsigned int height, unsigned int x, unsigned int y)</td>
<td>Updates the texture with rectangular pixels</td>
</tr>
<tr>
<td>update(const Image &amp;image)</td>
<td>Updates the texture with an Image</td>
</tr>
<tr>
<td>update(const Image &amp;image, unsigned int x, unsigned int y)</td>
<td>Updates the texture at a specific position</td>
</tr>
<tr>
<td>update(const Window &amp;window)</td>
<td>Updates the texture with a window</td>
</tr>
<tr>
<td>update(const Window &amp;window, unsigned int x, unsigned int y)</td>
<td>Updates the texture at a specific position</td>
</tr>
<tr>
<td>~GlResource()</td>
<td>Destroys the texture object</td>
</tr>
<tr>
<td>~Texture()</td>
<td>Destroys the texture</td>
</tr>
</tbody>
</table>
sf::Transform Member List

This is the complete list of members for sf::Transform, including all inherited members.

- `combine(const Transform &transform)`
- `getInverse() const`
- `getMatrix() const`
- `Identity`
- `operator*(const Transform &left, const Transform &right)`
- `operator*(const Transform &left, const Vector2f &right)`
- `operator*=(Transform &left, const Transform &right)`
- `rotate(float angle)`
- `rotate(float angle, float centerX, float centerY)`
- `rotate(float angle, const Vector2f &center)`
- `scale(float scaleX, float scaleY)`
- `scale(float scaleX, float scaleY, float centerX, float centerY)`
- `scale(const Vector2f &factors)`
- `scale(const Vector2f &factors, const Vector2f &center)`


Transform()

Transform(float a00, float a01, float a02, float a10, float a11, float a12, float a20, float a21, float a22)

transformPoint(float x, float y) const

transformPoint(const Vector2f &point) const

transformRect(const FloatRect &rectangle) const

translate(float x, float y)

translate(const Vector2f &offset)
The complete list of members for `sf::Transformable`, including all inherited members:

<table>
<thead>
<tr>
<th>Member Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getInverseTransform()</code> const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>getOrigin()</code> const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>getPosition()</code> const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>getRotation()</code> const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>getScale()</code> const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>getTransform()</code> const</td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>move(float offsetX, float offsetY)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>move(const Vector2f &amp;offset)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>rotate(float angle)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>scale(float factorX, float factorY)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>scale(const Vector2f &amp;factor)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setOrigin(float x, float y)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setOrigin(const Vector2f &amp;origin)</code></td>
<td>sf::Transformable</td>
</tr>
<tr>
<td><code>setPosition(float x, float y)</code></td>
<td>sf::Transformable</td>
</tr>
</tbody>
</table>
setPosition(const Vector2f &position)
setRotation(float angle)
setScale(float factorX, float factorY)
setScale(const Vector2f &factors)
Transformable()
~Transformable()

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Vertex Member List

This is the complete list of members for sf::Vertex, including all inherited members:

<table>
<thead>
<tr>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
</tr>
<tr>
<td>position</td>
</tr>
<tr>
<td>texCoords</td>
</tr>
<tr>
<td>Vertex()</td>
</tr>
<tr>
<td>Vertex(const Vector2f &amp;thePosition)</td>
</tr>
<tr>
<td>Vertex(const Vector2f &amp;thePosition, const Color &amp;theColor)</td>
</tr>
<tr>
<td>Vertex(const Vector2f &amp;thePosition, const Vector2f &amp;theTexCoords)</td>
</tr>
<tr>
<td>Vertex(const Vector2f &amp;thePosition, const Color &amp;theColor, const Vector2f &amp;theTexCoords)</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::VertexArray Member List

This is the complete list of members for sf::VertexArray, including all inherited members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append(const Vertex &amp;vertex)</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>clear()</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>getBounds() const</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>getPrimitiveType() const</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>getVertexCount() const</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>operator[](std::size_t index)</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>operator[](std::size_t index) const</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>resize(std::size_t vertexCount)</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>setPrimitiveType(PrimitiveType type)</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>VertexArray()</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>VertexArray(PrimitiveType type, std::size_t vertexCount=0)</td>
<td>sf::VertexArray</td>
</tr>
<tr>
<td>~Drawable()</td>
<td>sf::Drawable</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
sf::View Member List

This is the complete list of members for sf::View, including all inherited members:

- `getCenter() const` sf::View
- `getInverseTransform() const` sf::View
- `getRotation() const` sf::View
- `getSize() const` sf::View
- `getTransform() const` sf::View
- `getViewport() const` sf::View
- `move(float offsetX, float offsetY)` sf::View
- `move(const Vector2f &offset)` sf::View
- `reset(const FloatRect &rectangle)` sf::View
- `rotate(float angle)` sf::View
- `setCenter(float x, float y)` sf::View
- `setCenter(const Vector2f &center)` sf::View
- `setRotation(float angle)` sf::View
- `setSize(float width, float height)` sf::View
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setSize(const Vector2f &amp;size)</code></td>
<td>sets the size of the view</td>
</tr>
<tr>
<td><code>setViewport(const FloatRect &amp;viewport)</code></td>
<td>sets the viewport for the view</td>
</tr>
<tr>
<td><code>View()</code></td>
<td>creates a default view</td>
</tr>
<tr>
<td><code>View(const FloatRect &amp;rectangle)</code></td>
<td>creates a view with the given rectangle</td>
</tr>
<tr>
<td><code>View(const Vector2f &amp;center, const Vector2f &amp;size)</code></td>
<td>creates a view with the given center and size</td>
</tr>
<tr>
<td><code>zoom(float factor)</code></td>
<td>zooms the view by the given factor</td>
</tr>
</tbody>
</table>
sf::Ftp Member List

This is the complete list of members for sf::Ftp, including all inherited members:

- **Ascii** enum value
- **Binary** enum value
- **changeDirectory**(const std::string &directory)
- **connect**(const IpAddress &server, unsigned short port=21, Time timeout=)
- **createDirectory**(const std::string &name)
- **DataChannel** (defined in sf::Ftp)
- **deleteDirectory**(const std::string &name)
- **deleteFile**(const std::string &name)
- **disconnect**()
- **download**(const std::string &remoteFile, const std::string &localPath, TransferMode mode=Binary)
- **Ebcdic** enum value
- **getDirectoryListing**(const std::string &directory="")
- **getWorkingDirectory**()
- **keepAlive**()
login()

login(const std::string &name, const std::string &password)

NonCopyable()

parentDirectory()

renameFile(const std::string &file, const std::string &newName)

sendCommand(const std::string &command, const std::string &parameter=

TransferMode enum name

upload(const std::string &localFile, const std::string &remotePath, TransferMode mode=Binary)

~Ftp()
**sf::Ftp::DirectoryResponse Member List**

This is the complete list of members for `sf::Ftp::DirectoryResponse`, including all inherited members.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadCommandSequence enum value</td>
<td></td>
</tr>
<tr>
<td>ClosingConnection enum value</td>
<td></td>
</tr>
<tr>
<td>ClosingDataConnection enum value</td>
<td></td>
</tr>
<tr>
<td>CommandNotImplemented enum value</td>
<td></td>
</tr>
<tr>
<td>CommandUnknown enum value</td>
<td></td>
</tr>
<tr>
<td>ConnectionClosed enum value</td>
<td></td>
</tr>
<tr>
<td>ConnectionFailed enum value</td>
<td></td>
</tr>
<tr>
<td>DataConnectionAlreadyOpened enum value</td>
<td></td>
</tr>
<tr>
<td>DataConnectionOpened enum value</td>
<td></td>
</tr>
<tr>
<td>DataConnectionUnavailable enum value</td>
<td></td>
</tr>
<tr>
<td>DirectoryOk enum value</td>
<td></td>
</tr>
<tr>
<td>DirectoryResponse(const Response &amp;response)</td>
<td></td>
</tr>
<tr>
<td>DirectoryStatus enum value</td>
<td></td>
</tr>
<tr>
<td>EnteringPassiveMode enum value</td>
<td></td>
</tr>
<tr>
<td>Enum Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>FileActionAborted</td>
<td>Enum value</td>
</tr>
<tr>
<td>FileActionOk</td>
<td>Enum value</td>
</tr>
<tr>
<td>FilenameNotAllowed</td>
<td>Enum value</td>
</tr>
<tr>
<td>FileStatus</td>
<td>Enum value</td>
</tr>
<tr>
<td>FileUnavailable</td>
<td>Enum value</td>
</tr>
<tr>
<td>getDirectory() const</td>
<td></td>
</tr>
<tr>
<td>getMessage() const</td>
<td></td>
</tr>
<tr>
<td>getStatus() const</td>
<td></td>
</tr>
<tr>
<td>HelpMessage</td>
<td>Enum value</td>
</tr>
<tr>
<td>InsufficientStorageSpace</td>
<td>Enum value</td>
</tr>
<tr>
<td>InvalidFile</td>
<td>Enum value</td>
</tr>
<tr>
<td>InvalidResponse</td>
<td>Enum value</td>
</tr>
<tr>
<td>isOk() const</td>
<td></td>
</tr>
<tr>
<td>LocalError</td>
<td>Enum value</td>
</tr>
<tr>
<td>LoggedIn</td>
<td>Enum value</td>
</tr>
<tr>
<td>NeedAccountToLogIn</td>
<td>Enum value</td>
</tr>
<tr>
<td>NeedAccountToStore</td>
<td>Enum value</td>
</tr>
<tr>
<td>NeedInformation</td>
<td>Enum value</td>
</tr>
<tr>
<td>NeedPassword</td>
<td>Enum value</td>
</tr>
<tr>
<td>NotEnoughMemory</td>
<td>Enum value</td>
</tr>
<tr>
<td>NotLoggedIn</td>
<td>Enum value</td>
</tr>
<tr>
<td>Ok</td>
<td>Enum value</td>
</tr>
<tr>
<td>OpeningDataConnection</td>
<td>Enum value</td>
</tr>
<tr>
<td>PageTypeUnknown</td>
<td>Enum value</td>
</tr>
<tr>
<td>ParameterNotImplemented</td>
<td>Enum value</td>
</tr>
<tr>
<td>ParametersUnknown</td>
<td>Enum value</td>
</tr>
</tbody>
</table>
PointlessCommand enum value
Response(Status code=InvalidResponse, const std::string &message="")
RestartMarkerReply enum value
ServiceReady enum value
ServiceReadySoon enum value
ServiceUnavailable enum value
Status enum name
SystemStatus enum value
SystemType enum value
TransferAborted enum value
This is the complete list of members for `sf::Ftp::ListingResponse`, including all inherited members.

<table>
<thead>
<tr>
<th>Enum Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadCommandSequence</td>
<td>enum value</td>
</tr>
<tr>
<td>ClosingConnection</td>
<td>enum value</td>
</tr>
<tr>
<td>ClosingDataConnection</td>
<td>enum value</td>
</tr>
<tr>
<td>CommandNotImplemented</td>
<td>enum value</td>
</tr>
<tr>
<td>CommandUnknown</td>
<td>enum value</td>
</tr>
<tr>
<td>ConnectionClosed</td>
<td>enum value</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>enum value</td>
</tr>
<tr>
<td>DataConnectionAlreadyOpened</td>
<td>enum value</td>
</tr>
<tr>
<td>DataConnectionOpened</td>
<td>enum value</td>
</tr>
<tr>
<td>DataConnectionUnavailable</td>
<td>enum value</td>
</tr>
<tr>
<td>DirectoryOk</td>
<td>enum value</td>
</tr>
<tr>
<td>DirectoryStatus</td>
<td>enum value</td>
</tr>
<tr>
<td>EnteringPassiveMode</td>
<td>enum value</td>
</tr>
<tr>
<td>FileActionAborted</td>
<td>enum value</td>
</tr>
</tbody>
</table>
FileActionOk enum value
FilenameNotAllowed enum value
FileStatus enum value
FileUnavailable enum value
getListing() const
getMessage() const
getStatus() const
HelpMessage enum value
InsufficientStorageSpace enum value
InvalidFile enum value
InvalidResponse enum value
isOk() const
ListingResponse(const Response &response, const std::string &data)
LocalError enum value
LoggedIn enum value
NeedAccountToLogIn enum value
NeedAccountToStore enum value
NeedInformation enum value
NeedPassword enum value
NotEnoughMemory enum value
NotLoggedIN enum value
Ok enum value
OpeningDataConnection enum value
PageTypeUnknown enum value
ParameterNotImplemented enum value
ParametersUnknown enum value
<table>
<thead>
<tr>
<th>Enum/Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointlessCommand</td>
<td>Enum value</td>
</tr>
<tr>
<td>Response</td>
<td>(Status code=InvalidResponse, const std::string &amp;message=&quot;&quot;)</td>
</tr>
<tr>
<td>RestartMarkerReply</td>
<td>Enum value</td>
</tr>
<tr>
<td>ServiceReady</td>
<td>Enum value</td>
</tr>
<tr>
<td>ServiceReadySoon</td>
<td>Enum value</td>
</tr>
<tr>
<td>ServiceUnavailable</td>
<td>Enum value</td>
</tr>
<tr>
<td>Status</td>
<td>Enum name</td>
</tr>
<tr>
<td>SystemStatus</td>
<td>Enum value</td>
</tr>
<tr>
<td>SystemType</td>
<td>Enum value</td>
</tr>
<tr>
<td>TransferAborted</td>
<td>Enum value</td>
</tr>
</tbody>
</table>
sf::Ftp::Response Member List

This is the complete list of members for sf::Ftp::Response, including all inherited members.

- **BadCommandSequence** enum value
- **ClosingConnection** enum value
- **ClosingDataConnection** enum value
- **CommandNotImplemented** enum value
- **CommandUnknown** enum value
- **ConnectionClosed** enum value
- **ConnectionFailed** enum value
- **DataConnectionAlreadyOpened** enum value
- **DataConnectionOpened** enum value
- **DataConnectionUnavailable** enum value
- **DirectoryOk** enum value
- **DirectoryStatus** enum value
- **EnteringPassiveMode** enum value
- **FileActionAborted** enum value
enum FileActionOk
enum FilenameNotAllowed
enum FileStatus
enum FileUnavailable

const std::string getMessage()
const std::string getStatus()

enum HelpMessage
enum InsufficientStorageSpace
enum InvalidFile
enum InvalidResponse

const std::string isOk()

enum LocalError
enum LoggedIn

enum NeedAccountToLogIn
enum NeedAccountToStore
enum NeedInformation
enum NeedPassword

enum NotEnoughMemory
enum NotLoggedIn

enum Ok
enum OpeningDataConnection
enum PageTypeUnknown

enum ParameterNotImplemented
enum ParametersUnknown
enum PointlessCommand

Response(Status code=InvalidResponse, const std::string &message="")
RestartMarkerReply enum value
ServiceReady enum value
ServiceReadySoon enum value
ServiceUnavailable enum value
Status enum name
SystemStatus enum value
SystemType enum value
TransferAborted enum value
### sf::Http Member List

This is the complete list of members for `sf::Http`, including all inherited members.

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Http()</code></td>
<td>sf::Http</td>
</tr>
<tr>
<td><code>Http(const std::string &amp;host, unsigned short port=0)</code></td>
<td>sf::Http</td>
</tr>
<tr>
<td><code>NonCopyable()</code></td>
<td>sf::Non</td>
</tr>
<tr>
<td><code>sendRequest(const Request &amp;request, Time timeout=Time::Zero)</code></td>
<td>sf::Http</td>
</tr>
<tr>
<td><code>setHost(const std::string &amp;host, unsigned short port=0)</code></td>
<td>sf::Http</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
**sf::Http::Request Member List**

This is the complete list of members for `sf::Http::Request`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>enum value</td>
</tr>
<tr>
<td>Get</td>
<td>enum value</td>
</tr>
<tr>
<td>Head</td>
<td>enum value</td>
</tr>
<tr>
<td><strong>Http</strong></td>
<td>(defined in <code>sf::Http::Request</code>)</td>
</tr>
<tr>
<td>Method</td>
<td>enum name</td>
</tr>
<tr>
<td>Post</td>
<td>enum value</td>
</tr>
<tr>
<td>Put</td>
<td>enum value</td>
</tr>
<tr>
<td>Request</td>
<td>(const std::string &amp;uri=&quot;/&quot;, Method method=Get, const std::string &amp;body=&quot;&quot;)</td>
</tr>
<tr>
<td>setBody</td>
<td>(const std::string &amp;body)</td>
</tr>
<tr>
<td>setField</td>
<td>(const std::string &amp;field, const std::string &amp;value)</td>
</tr>
<tr>
<td>setHttpVersion</td>
<td>(unsigned int major, unsigned int minor)</td>
</tr>
<tr>
<td>setMethod</td>
<td>(Method method)</td>
</tr>
<tr>
<td>setUri</td>
<td>(const std::string &amp;uri)</td>
</tr>
</tbody>
</table>
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
### sf::Http::Response Member List

This is the complete list of members for `sf::Http::Response`, including all its enum values and functions.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accepted</td>
<td>enum value</td>
</tr>
<tr>
<td>BadGateway</td>
<td>enum value</td>
</tr>
<tr>
<td>BadRequest</td>
<td>enum value</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>enum value</td>
</tr>
<tr>
<td>Created</td>
<td>enum value</td>
</tr>
<tr>
<td>Forbidden</td>
<td>enum value</td>
</tr>
<tr>
<td>GatewayTimeout</td>
<td>enum value</td>
</tr>
<tr>
<td>getBody() const</td>
<td></td>
</tr>
<tr>
<td>getField(const std::string &amp;field) const</td>
<td></td>
</tr>
<tr>
<td>getMajorHttpVersion() const</td>
<td></td>
</tr>
<tr>
<td>getMinorHttpVersion() const</td>
<td></td>
</tr>
<tr>
<td>getStatus() const</td>
<td></td>
</tr>
<tr>
<td><strong>Http</strong> <em>(defined in sf::Http::Response)</em></td>
<td></td>
</tr>
<tr>
<td>InternalServerError</td>
<td>enum value</td>
</tr>
<tr>
<td>Enum Name</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>InvalidResponse</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>MovedPermanently</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>MovedTemporarily</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>MultipleChoices</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>NoContent</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>NotFound</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>NotImplemented</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>NotModified</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>Ok</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>PartialContent</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>RangeNotSatisfiable</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>ResetContent</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>Response</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>ServiceNotAvailable</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>Status</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>Unauthorized</td>
<td>sf::Http::Response</td>
</tr>
<tr>
<td>VersionNotSupported</td>
<td>sf::Http::Response</td>
</tr>
</tbody>
</table>
sf::IpAddress Member List

This is the complete list of members for `sf::IpAddress`, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>Broadcast</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>getLocalAddress()</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>getPublicAddress(Time timeout=Time::Zero)</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>IpAddress()</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>IpAddress(const std::string &amp;address)</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>IpAddress(const char *address)</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>IpAddress(Uint8 byte0, Uint8 byte1, Uint8 byte2, Uint8 byte3)</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>IpAddress(Uint32 address)</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>LocalHost</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>None</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>operator&lt;(const IpAddress &amp;left, const IpAddress &amp;right)</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>toInteger() const</td>
<td>sf::IpAddress</td>
</tr>
<tr>
<td>toString() const</td>
<td>sf::IpAddress</td>
</tr>
</tbody>
</table>
This is the complete list of members for `sf::Packet`, including all inherited members:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>append(const void *data, std::size_t sizeInBytes)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>endOfPacket()</code> const</td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>getData()</code> const</td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>getDataSize()</code> const</td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>onReceive(const void *data, std::size_t size)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>onSend(std::size_t &amp;size)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>operator BoolType() const</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(bool data)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Int8 data)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Uint8 data)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Int16 data)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Uint16 data)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Int32 data)</code></td>
<td>sf::Packet</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Uint32 data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Int64 data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(Uint64 data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(float data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(double data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(const char *data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(const std::string &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(const wchar_t *data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(const std::wstring &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&lt;&lt;(const String &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(bool &amp;data)</code></td>
<td></td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Int8 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Uint8 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Int16 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Uint16 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Int32 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Uint32 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Int64 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(Uint64 &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(float &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(double &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(char *data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(std::string &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(wchar_t *data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(std::wstring &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td><code>operator&gt;&gt;(String &amp;data)</code></td>
<td>(defined in <code>sf::Packet</code>)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Packet()</td>
<td></td>
</tr>
<tr>
<td>TcpSocket (defined in sf::Packet)</td>
<td>sf::Packet</td>
</tr>
<tr>
<td>UdpSocket (defined in sf::Packet)</td>
<td>sf::Packet</td>
</tr>
<tr>
<td>~Packet()</td>
<td></td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila  ::  Documentation generated by doxygen
sf::Socket Member List

This is the complete list of members for sf::Socket, including all inherited members.

<table>
<thead>
<tr>
<th>AnyPort enum value</th>
<th>sf::Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>close()</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>create()</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>create(SocketHandle handle)</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>Disconnected enum value</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>Done enum value</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>Error enum value</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>getHandle() const</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>isBlocking() const</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>NonCopyable()</td>
<td>sf::NonCopyable</td>
</tr>
<tr>
<td>NotReady enum value</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>Partial enum value</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>setBlocking(bool blocking)</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>Socket(Type type)</td>
<td>sf::Socket</td>
</tr>
<tr>
<td><strong>SocketSelector</strong> (defined in sf::Socket)</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>Status</strong> enum name</td>
<td>sf::Socket</td>
</tr>
<tr>
<td><strong>Tcp</strong> enum value</td>
<td>sf::Socket</td>
</tr>
<tr>
<td><strong>Type</strong> enum name</td>
<td>sf::Socket</td>
</tr>
<tr>
<td><strong>Udp</strong> enum value</td>
<td>sf::Socket</td>
</tr>
<tr>
<td>~Socket()</td>
<td>sf::Socket</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::SocketSelector Member List

This is the complete list of members for sf::SocketSelector, including all in

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(Socket &amp;socket)</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>clear()</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>isReady(Socket &amp;socket) const</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>operator=(const SocketSelector &amp;right)</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>remove(Socket &amp;socket)</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>SocketSelector()</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>SocketSelector(const SocketSelector &amp;copy)</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>wait(Time timeout=Time::Zero)</td>
<td>sf::SocketSelector</td>
</tr>
<tr>
<td>~SocketSelector()</td>
<td>sf::SocketSelector</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
This is the complete list of members for `sf::TcpListener`, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>accept(TcpSocket &amp;socket)</code></td>
<td>Accepts a connection from a client.</td>
</tr>
<tr>
<td><code>AnyPort enum value</code></td>
<td>Represents any port.</td>
</tr>
<tr>
<td><code>close()</code></td>
<td>Closes the socket.</td>
</tr>
<tr>
<td><code>create()</code></td>
<td>Creates a new socket.</td>
</tr>
<tr>
<td><code>create(SocketHandle handle)</code></td>
<td>Creates a new socket with a specific handle.</td>
</tr>
<tr>
<td><code>Disconnected enum value</code></td>
<td>Indicates the disconnected state.</td>
</tr>
<tr>
<td><code>Done enum value</code></td>
<td>Represents the done state.</td>
</tr>
<tr>
<td><code>Error enum value</code></td>
<td>Represents an error.</td>
</tr>
<tr>
<td><code>getHandle() const</code></td>
<td>Gets the handle of the socket.</td>
</tr>
<tr>
<td><code>getLocalPort() const</code></td>
<td>Gets the local port of the socket.</td>
</tr>
<tr>
<td><code>isBlocking() const</code></td>
<td>Indicates if the socket is blocking or not.</td>
</tr>
<tr>
<td><code>listen(unsigned short port, const IpAddress &amp;address=IpAddress::Any)</code></td>
<td>Listens for incoming connections on a specific port.</td>
</tr>
<tr>
<td><code>NotReady enum value</code></td>
<td>Indicates the not ready state.</td>
</tr>
<tr>
<td><code>Partial enum value</code></td>
<td>Represents a partial state.</td>
</tr>
<tr>
<td>Function/Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>setBlocking</code></td>
<td><code>bool blocking</code></td>
</tr>
<tr>
<td><code>Socket</code></td>
<td><code>Type type</code></td>
</tr>
<tr>
<td><code>Status</code></td>
<td><code>enum name</code></td>
</tr>
<tr>
<td><code>Tcp</code></td>
<td><code>enum value</code></td>
</tr>
<tr>
<td><code>TcpListener()</code></td>
<td></td>
</tr>
<tr>
<td><code>Type</code></td>
<td><code>enum name</code></td>
</tr>
<tr>
<td><code>Udp</code></td>
<td><code>enum value</code></td>
</tr>
<tr>
<td><code>~Socket()</code></td>
<td></td>
</tr>
</tbody>
</table>
sf::TcpSocket Member List

This is the complete list of members for sf::TcpSocket, including all inherited members.

<table>
<thead>
<tr>
<th>AnyPort enum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>close()</td>
</tr>
<tr>
<td>connect(const IpAddress &amp;remoteAddress, unsigned short remotePort, Time timeout=Time::Zero)</td>
</tr>
<tr>
<td>create()</td>
</tr>
<tr>
<td>create(SocketHandle handle)</td>
</tr>
<tr>
<td>disconnect()</td>
</tr>
<tr>
<td>Disconnected enum value</td>
</tr>
<tr>
<td>Done enum value</td>
</tr>
<tr>
<td>Error enum value</td>
</tr>
<tr>
<td>getHandle() const</td>
</tr>
<tr>
<td>getLocalPort() const</td>
</tr>
<tr>
<td>getRemoteAddress() const</td>
</tr>
<tr>
<td>getRemotePort() const</td>
</tr>
<tr>
<td>isBlocking() const</td>
</tr>
</tbody>
</table>
### NotReady enum value

### Partial enum value

```cpp
receive(void *data, std::size_t size, std::size_t &received)
```

```cpp
receive(Packet &packet)
```

```cpp
send(const void *data, std::size_t size)
```

```cpp
send(const void *data, std::size_t size, std::size_t &sent)
```

```cpp
send(Packet &packet)
```

```cpp
setBlocking(bool blocking)
```

```cpp
Socket(Type type)
```

```cpp
Status enum name
```

### Tcp enum value

#### TcpListener (defined in sf::TcpSocket)

```cpp
TcpSocket()
```

### Type enum name

### Udp enum value

```cpp
~Socket()
```
sf::UdpSocket Member List

This is the complete list of members for sf::UdpSocket, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnyPort</td>
<td>enum value</td>
</tr>
<tr>
<td>bind</td>
<td>(unsigned short port, const IpAddress &amp;address=IpAddress::Any)</td>
</tr>
<tr>
<td>close</td>
<td>()</td>
</tr>
<tr>
<td>create</td>
<td>()</td>
</tr>
<tr>
<td>create</td>
<td>(SocketHandle handle)</td>
</tr>
<tr>
<td>Disconnected</td>
<td>enum value</td>
</tr>
<tr>
<td>Done</td>
<td>enum value</td>
</tr>
<tr>
<td>Error</td>
<td>enum value</td>
</tr>
<tr>
<td>getHandle</td>
<td>() const</td>
</tr>
<tr>
<td>getLocalPort</td>
<td>() const</td>
</tr>
<tr>
<td>isBlocking</td>
<td>() const</td>
</tr>
<tr>
<td>MaxDatagramSize</td>
<td>enum value</td>
</tr>
<tr>
<td>NotReady</td>
<td>enum value</td>
</tr>
<tr>
<td>Partial</td>
<td>enum value</td>
</tr>
</tbody>
</table>
receive (void *data, std::size_t size, std::size_t &received, IpAddress &remoteAddress, unsigned short &remotePort)

receive (Packet &packet, IpAddress &remoteAddress, unsigned short &remotePort)

send (const void *data, std::size_t size, const IpAddress &remoteAddress, unsigned short remotePort)

send (Packet &packet, const IpAddress &remoteAddress, unsigned short remotePort)

setBlocking (bool blocking)

Socket (Type type)

Status enum name

Tcp enum value

Type enum name

Udp enum value

UdpSocket()

unbind()

~Socket()
sf::Clock Member List

This is the complete list of members for sf::Clock, including all inherited members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf::Clock</td>
<td>sf::Clock()</td>
</tr>
<tr>
<td>sf::Clock</td>
<td>getElapsedTime() const</td>
</tr>
<tr>
<td>sf::Clock</td>
<td>restart()</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.  
Copyright © Laurent Gomila  ::  Documentation generated by doxygen.
### sf::FileInputStream Member List

This is the complete list of members for `sf::FileInputStream`, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FileInputStream()</code></td>
<td><code>sf::FileInputStream</code></td>
</tr>
<tr>
<td><code>getSize()</code></td>
<td><code>sf::FileInputStream virtual</code></td>
</tr>
<tr>
<td><code>NonCopyable()</code></td>
<td><code>sf::NonCopyable inline private</code></td>
</tr>
<tr>
<td><code>open(const std::string &amp;filename)</code></td>
<td><code>sf::FileInputStream</code></td>
</tr>
<tr>
<td><code>read(void *data, Int64 size)</code></td>
<td><code>sf::FileInputStream virtual</code></td>
</tr>
<tr>
<td><code>seek(Int64 position)</code></td>
<td><code>sf::FileInputStream virtual</code></td>
</tr>
<tr>
<td><code>tell()</code></td>
<td><code>sf::FileInputStream virtual</code></td>
</tr>
<tr>
<td><code>~FileInputStream()</code></td>
<td><code>sf::FileInputStream virtual</code></td>
</tr>
<tr>
<td><code>~InputStream()</code></td>
<td><code>sf::InputStream inline virtual</code></td>
</tr>
</tbody>
</table>
sf::InputStream Member List

This is the complete list of members for sf::InputStream, including all inherited members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSize()=0</td>
<td>sf::InputStream pure virtual</td>
</tr>
<tr>
<td>read(void *data, Int64 size)=0</td>
<td>sf::InputStream pure virtual</td>
</tr>
<tr>
<td>seek(Int64 position)=0</td>
<td>sf::InputStream pure virtual</td>
</tr>
<tr>
<td>tell()=0</td>
<td>sf::InputStream pure virtual</td>
</tr>
<tr>
<td>~InputStream()</td>
<td>sf::InputStream inline virtual</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Lock Member List

This is the complete list of members for sf::Lock, including all inherited members:

<table>
<thead>
<tr>
<th>Lock(Mutex &amp;mutex) sf::Lock</th>
<th>sf::NonCopyable inline private</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonCopyable() sf::NonCopyable</td>
<td>sf::Lock</td>
</tr>
<tr>
<td>~Lock()                     sf::Lock</td>
<td></td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::MemoryInputStream Member List

This is the complete list of members for sf::MemoryInputStream, including:

- `getSize()` sf::MemoryInputStream
- `MemoryInputStream()` sf::MemoryInputStream
- `open(const void *data, std::size_t sizeInBytes)` sf::MemoryInputStream
- `read(void *data, Int64 size)` sf::MemoryInputStream
- `seek(Int64 position)` sf::MemoryInputStream
- `tell()` sf::MemoryInputStream
- `~InputStream()` sf::InputStream

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Mutex Member List

This is the complete list of members for sf::Mutex, including all inherited members:

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock()</td>
<td>sf::Mutex</td>
</tr>
<tr>
<td>Mutex()</td>
<td>sf::Mutex</td>
</tr>
<tr>
<td>NonCopyable()</td>
<td>sf::NonCopyable</td>
</tr>
<tr>
<td>unlock()</td>
<td>sf::Mutex</td>
</tr>
<tr>
<td>~Mutex()</td>
<td>sf::Mutex</td>
</tr>
</tbody>
</table>
sf::NonCopyable Member List

This is the complete list of members for sf::NonCopyable, including all inh

NonCopyable() sf::NonCopyable inline protected
sf::String Member List

This is the complete list of members for sf::String, including all inherited members:

- `begin()`
- `begin() const`
- `clear()`
- `ConstIterator typedef`
- `end()`
- `end() const`
- `erase(std::size_t position, std::size_t count=1)`
- `find(const String &str, std::size_t start=0) const`
- `fromUtf16(T begin, T end)`
- `fromUtf32(T begin, T end)`
- `fromUtf8(T begin, T end)`
- `getData() const`
- `getSize() const`
- `insert(std::size_t position, const String &str)`
InvalidPos
isEmpty() const
Iterator typedef
operator std::string() const
operator std::wstring() const
operator!=(const String &left, const String &right)
operator+(const String &left, const String &right)
operator+=(const String &right)
operator<(defined in sf::String)
operator<(const String &left, const String &right)
operator<=(const String &left, const String &right)
operator=(const String &right)
operator==(defined in sf::String)
operator==(const String &left, const String &right)
operator>(const String &left, const String &right)
operator>=(const String &left, const String &right)
operator[](std::size_t index) const
operator[](std::size_t index)
replace(std::size_t position, std::size_t length, const String &replaceWith)
replace(const String &searchFor, const String &replaceWith)
String()
String(char ansiChar, const std::locale &locale=std::locale())
String(wchar_t wideChar)
String(Uint32 utf32Char)
String(const char *ansiString, const std::locale &locale=std::locale())
String(const std::string &ansiString, const std::locale &locale=std::locale())
String(const wchar_t *wideString)
String(const std::wstring &wideString)
String(const Uint32 *utf32String)
String(const std::basic_string< Uint32 > &utf32String)
String(const String &copy)

substring(std::size_t position, std::size_t length=InvalidPos) const

toAnsiString(const std::locale &locale=std::locale()) const

toUtf16() const

toUtf32() const

toUtf8() const

toWideString() const
## sf::Thread Member List

This is the complete list of members for `sf::Thread`, including all inherited members:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>launch()</code></td>
<td><code>sf::Thread</code></td>
</tr>
<tr>
<td><code>NonCopyable()</code></td>
<td><code>sf::NonCopyable inline private</code></td>
</tr>
<tr>
<td><code>terminate()</code></td>
<td><code>sf::Thread</code></td>
</tr>
<tr>
<td><code>Thread(F function)</code></td>
<td><code>sf::Thread</code></td>
</tr>
<tr>
<td><code>Thread(F function, A argument)</code></td>
<td><code>sf::Thread</code></td>
</tr>
<tr>
<td><code>Thread(void(C::*function)(), C *object)</code></td>
<td><code>sf::Thread</code></td>
</tr>
<tr>
<td><code>wait()</code></td>
<td><code>sf::Thread</code></td>
</tr>
<tr>
<td><code>~Thread()</code></td>
<td><code>sf::Thread</code></td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::ThreadLocal Member List

This is the complete list of members for sf::ThreadLocal, including all inhe

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValue() const</td>
<td>sf::ThreadLocal</td>
</tr>
<tr>
<td>NonCopyable()</td>
<td>sf::NonCopyable</td>
</tr>
<tr>
<td>setValue(void *value)</td>
<td>sf::ThreadLocal</td>
</tr>
<tr>
<td>ThreadLocal(void *value=NULL)</td>
<td>sf::ThreadLocal</td>
</tr>
<tr>
<td>~ThreadLocal()</td>
<td>sf::ThreadLocal</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::ThreadLocalPtr< T > Member List

This is the complete list of members for sf::ThreadLocalPtr< T >, including:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValue() const</td>
<td>sf::ThreadLocal</td>
</tr>
<tr>
<td>operator T *() const</td>
<td>sf::ThreadLocalPtr&lt; T &gt;</td>
</tr>
<tr>
<td>operator*() const</td>
<td>sf::ThreadLocalPtr&lt; T &gt;</td>
</tr>
<tr>
<td>operator-&gt;() const</td>
<td>sf::ThreadLocalPtr&lt; T &gt;</td>
</tr>
<tr>
<td>operator=(T *value)</td>
<td>sf::ThreadLocalPtr&lt; T &gt;</td>
</tr>
<tr>
<td>operator=(const ThreadLocalPtr&lt; T &gt; &amp;right)</td>
<td>sf::ThreadLocalPtr&lt; T &gt;</td>
</tr>
<tr>
<td>setValue(void *value)</td>
<td>sf::ThreadLocal</td>
</tr>
<tr>
<td>ThreadLocal(void *value=NULL)</td>
<td>sf::ThreadLocal</td>
</tr>
<tr>
<td>ThreadLocalPtr(T *value=NULL)</td>
<td>sf::ThreadLocalPtr&lt; T &gt;</td>
</tr>
<tr>
<td>~ThreadLocal()</td>
<td>sf::ThreadLocal</td>
</tr>
</tbody>
</table>
sf::Time Member List

This is the complete list of members for sf::Time, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asMicroseconds() const</td>
<td>sf::Time</td>
</tr>
<tr>
<td>asMilliseconds() const</td>
<td>sf::Time</td>
</tr>
<tr>
<td>asSeconds() const</td>
<td>sf::Time</td>
</tr>
<tr>
<td>microseconds (defined in sf::Time)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>microseconds(Int64 amount)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>milliseconds (defined in sf::Time)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>milliseconds(Int32 amount)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>operator!=(Time left, Time right)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>operator%(Time left, Time right)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>operator%(Time &amp;left, Time right)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>operator*(Time left, float right)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>operator*(Time left, Int64 right)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>operator*(float left, Time right)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>operator*(Int64 left, Time right)</td>
<td>sf::Time</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>*=</td>
<td>(Time &amp;left, float right)</td>
</tr>
<tr>
<td>*=</td>
<td>(Time &amp;left, Int64 right)</td>
</tr>
<tr>
<td>+=</td>
<td>(Time &amp;left, Time right)</td>
</tr>
<tr>
<td>-=</td>
<td>(Time right)</td>
</tr>
<tr>
<td>-=</td>
<td>(Time left, Time right)</td>
</tr>
<tr>
<td>-=</td>
<td>(Time &amp;left, Time right)</td>
</tr>
<tr>
<td>/=</td>
<td>(Time left, float right)</td>
</tr>
<tr>
<td>/=</td>
<td>(Time left, Int64 right)</td>
</tr>
<tr>
<td>/=</td>
<td>(Time left, Time right)</td>
</tr>
<tr>
<td>/=</td>
<td>(Time &amp;left, float right)</td>
</tr>
<tr>
<td>/=</td>
<td>(Time &amp;left, Int64 right)</td>
</tr>
<tr>
<td>&lt;</td>
<td>(Time left, Time right)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>(Time left, Time right)</td>
</tr>
<tr>
<td>==</td>
<td>(Time left, Time right)</td>
</tr>
<tr>
<td>&gt;</td>
<td>(Time left, Time right)</td>
</tr>
<tr>
<td>&gt;=</td>
<td>(Time left, Time right)</td>
</tr>
<tr>
<td>seconds</td>
<td>(defined in sf::Time)</td>
</tr>
<tr>
<td>seconds</td>
<td>(float amount)</td>
</tr>
<tr>
<td>Time()</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td></td>
</tr>
</tbody>
</table>
sf::Vector2< T > Member List

This is the complete list of members for sf::Vector2< T >, including all inherited members:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator!=(const Vector2&lt; T &gt; &amp;left, const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator*(const Vector2&lt; T &gt; &amp;left, T right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator*(T left, const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator*=(Vector2&lt; T &gt; &amp;left, T right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator+(const Vector2&lt; T &gt; &amp;left, const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator+=(Vector2&lt; T &gt; &amp;left, const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator-(const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator-(const Vector2&lt; T &gt; &amp;left, const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator-=(Vector2&lt; T &gt; &amp;left, const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator/((const Vector2&lt; T &gt; &amp;left, T right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator/=(Vector2&lt; T &gt; &amp;left, T right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>operator==(const Vector2&lt; T &gt; &amp;left, const Vector2&lt; T &gt; &amp;right)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>Vector2()</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>Vector2(T X, T Y)</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>Vector2(const Vector2&lt; U &gt; &amp;vector)</td>
<td>sf::Vector2&lt; U &gt;</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>x</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
<tr>
<td>y</td>
<td>sf::Vector2&lt; T &gt;</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila  ::  Documentation generated by doxygen
### sf::Vector3< T > Member List

This is the complete list of members for `sf::Vector3< T >`, including all inherited members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator!=(const Vector3&lt; T &gt;&amp;left, const Vector3&lt; T &gt;&amp;right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator*(const Vector3&lt; T &gt;&amp;left, T right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator*(T left, const Vector3&lt; T &gt;&amp;right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator*=(Vector3&lt; T &gt;&amp;left, T right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator+(const Vector3&lt; T &gt;&amp;left, const Vector3&lt; T &gt;&amp;right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator+=(Vector3&lt; T &gt;&amp;left, const Vector3&lt; T &gt;&amp;right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator-=(Vector3&lt; T &gt;&amp;left, const Vector3&lt; T &gt;&amp;right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator/=(const Vector3&lt; T &gt;&amp;left, T right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator/=(Vector3&lt; T &gt;&amp;left, T right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
<tr>
<td><code>operator==(const Vector3&lt; T &gt;&amp;left, const Vector3&lt; T &gt;&amp;right)</code></td>
<td><code>sf::Vector3&lt; T &gt;</code></td>
</tr>
</tbody>
</table>

#### Constructors

- `Vector3()`
- `Vector3(T X, T Y, T Z)`
<table>
<thead>
<tr>
<th>Vector3(const Vector3&lt; U &gt; &amp;vector)</th>
<th>sf::Vector3&lt; T &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>sf::Vector3&lt; T &gt;</td>
</tr>
<tr>
<td>y</td>
<td>sf::Vector3&lt; T &gt;</td>
</tr>
<tr>
<td>z</td>
<td>sf::Vector3&lt; T &gt;</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila  ::  Documentation generated by doxygen
sf::Context Member List

This is the complete list of members for sf::Context, including all inherited

- Context()
- Context(const ContextSettings &settings, unsigned int width, unsigned int height)
- ensureGIContext()
- getActiveContext()
- getFunction(const char *name)
- getSettings() const
- GlResource()
- isExtensionAvailable(const char *name)
- NonCopyable()
- setActive(bool active)
- ~Context()
- ~GlResource()

SFML is licensed under the terms and conditions of the zlib/png license
sf::ContextSettings Member List

This is the complete list of members for sf::ContextSettings, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>antialiasingLevel</td>
<td>Attribute enum name</td>
</tr>
<tr>
<td>attributeFlags</td>
<td>ContextSettings(ulong int depth=0, unsigned int stencil=0, unsigned int antialiasing=0, unsigned int major=1, unsigned int minor=1, unsigned int attributes=Default, bool sRgb=false)</td>
</tr>
<tr>
<td>Core</td>
<td>Debug enum value</td>
</tr>
<tr>
<td>Default</td>
<td>Debug enum value</td>
</tr>
<tr>
<td>depthBits</td>
<td>Core enum value</td>
</tr>
<tr>
<td>majorVersion</td>
<td>Default enum value</td>
</tr>
<tr>
<td>minorVersion</td>
<td>depthBits</td>
</tr>
<tr>
<td>sRgbCapable</td>
<td>majorVersion</td>
</tr>
<tr>
<td>stencilBits</td>
<td>sRgbCapable</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>Count</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>EventType</td>
<td>enum name</td>
<td>sf::Event</td>
</tr>
<tr>
<td>GainedFocus</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>joystickButton</td>
<td></td>
<td>sf::Event</td>
</tr>
<tr>
<td>JoystickButtonPressed</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>JoystickButtonReleased</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>joystickConnect</td>
<td></td>
<td>sf::Event</td>
</tr>
<tr>
<td>JoystickConnected</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>JoystickDisconnected</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>joystickMove</td>
<td></td>
<td>sf::Event</td>
</tr>
<tr>
<td>JoystickMoved</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>key</td>
<td></td>
<td>sf::Event</td>
</tr>
<tr>
<td>KeyPressed</td>
<td>enum value</td>
<td>sf::Event</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------</td>
<td></td>
</tr>
<tr>
<td>KeyReleased</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>LostFocus</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>mouseButton</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>MouseButtonPressed</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>MouseButtonReleased</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>MouseEntered</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>MouseLeft</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>mouseMove</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>MouseMoved</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>mouseWheel</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>MouseWheelMoved</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>mouseWheelScroll</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>MouseWheelScrolled</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>Resized</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>sensor</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>SensorChanged</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>TextEntered</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>touch</td>
<td>sf::Event</td>
<td></td>
</tr>
<tr>
<td>TouchBegan</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>TouchEnded</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>TouchMoved</td>
<td>enum value</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>sf::Event</td>
<td></td>
</tr>
</tbody>
</table>
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Event::JoystickButtonEvent Member List

This is the complete list of members for sf::Event::JoystickButtonEvent, in alphabetical order:

- button
- joystickId
sf::Event::JoystickConnectEvent Member List

This is the complete list of members for sf::Event::JoystickConnectEvent,

| joystickId | sf::Event::JoystickConnectEvent |

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Event::JoystickMoveEvent Member List

This is the complete list of members for sf::Event::JoystickMoveEvent, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>sf::Event::JoystickMoveEvent</td>
</tr>
<tr>
<td>joystickId</td>
<td>sf::Event::JoystickMoveEvent</td>
</tr>
<tr>
<td>position</td>
<td>sf::Event::JoystickMoveEvent</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Event::KeyEvent Member List

This is the complete list of members for sf::Event::KeyEvent, including all

<table>
<thead>
<tr>
<th></th>
<th>sf::Event::KeyEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt</td>
<td>sf::Event::KeyEvent</td>
</tr>
<tr>
<td>code</td>
<td>sf::Event::KeyEvent</td>
</tr>
<tr>
<td>control</td>
<td>sf::Event::KeyEvent</td>
</tr>
<tr>
<td>shift</td>
<td>sf::Event::KeyEvent</td>
</tr>
<tr>
<td>system</td>
<td>sf::Event::KeyEvent</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Event::MouseButtonEvent Member List

This is the complete list of members for sf::Event::MouseButtonEvent, including all inherited members.

- `button` sf::Event::MouseButtonEvent
- `x` sf::Event::MouseButtonEvent
- `y` sf::Event::MouseButtonEvent

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Event::MouseMoveEvent Member List

This is the complete list of members for sf::Event::MouseMoveEvent, including:

- x sf::Event::MouseMoveEvent
- y sf::Event::MouseMoveEvent
sf::Event::MouseWheelEvent Member List

This is the complete list of members for sf::Event::MouseWheelEvent, incl

- delta sf::Event::MouseWheelEvent
- x sf::Event::MouseWheelEvent
- y sf::Event::MouseWheelEvent

SFML is licensed under the terms and conditions of the zlib/png license. Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Event::MouseWheelScrollEvent Member List

This is the complete list of members for sf::Event::MouseWheelScrollEvent members.

<table>
<thead>
<tr>
<th>delta</th>
<th>sf::Event::MouseWheelScrollEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>wheel</td>
<td>sf::Event::MouseWheelScrollEvent</td>
</tr>
<tr>
<td>x</td>
<td>sf::Event::MouseWheelScrollEvent</td>
</tr>
<tr>
<td>y</td>
<td>sf::Event::MouseWheelScrollEvent</td>
</tr>
</tbody>
</table>
sf::Event::SensorEvent Member List

This is the complete list of members for sf::Event::SensorEvent, including:

- `type sf::Event::SensorEvent`
- `x sf::Event::SensorEvent`
- `y sf::Event::SensorEvent`
- `z sf::Event::SensorEvent`
sf::Event::SizeEvent Member List

This is the complete list of members for sf::Event::SizeEvent, including all

height  sf::Event::SizeEvent
width   sf::Event::SizeEvent
sf::Event::TextEvent Member List

This is the complete list of members for sf::Event::TextEvent, including all

unicode sf::Event::TextEvent

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Event::TouchEvent Member List

This is the complete list of members for sf::Event::TouchEvent, including:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>finger</td>
<td>sf::Event::TouchEvent</td>
</tr>
<tr>
<td>x</td>
<td>sf::Event::TouchEvent</td>
</tr>
<tr>
<td>y</td>
<td>sf::Event::TouchEvent</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::GlResource Member List

This is the complete list of members for sf::GlResource, including all inherited members:

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Access</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>ensureGlContext()</td>
<td>sf::GlResource</td>
<td>protected</td>
<td>static</td>
</tr>
<tr>
<td>GlResource()</td>
<td>sf::GlResource</td>
<td>protected</td>
<td></td>
</tr>
<tr>
<td>~GlResource()</td>
<td>sf::GlResource</td>
<td>protected</td>
<td></td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila. :: Documentation generated by doxygen.
sf::GlResource::TransientContextLock Members

This is the complete list of members for sf::GlResource::TransientContextLock members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonCopyable()</td>
<td>sf::NonCopyable</td>
</tr>
<tr>
<td>TransientContextLock()</td>
<td>sf::GlResource::TransientContextLock</td>
</tr>
<tr>
<td>~TransientContextLock()</td>
<td>sf::GlResource::TransientContextLock</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen
This is the complete list of members for `sf::Joystick`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis enum name</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>AxisCount enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>ButtonCount enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>Count enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>getAxisPosition</td>
<td>(unsigned int joystick, Axis axis)</td>
</tr>
<tr>
<td>getButtonCount</td>
<td>(unsigned int joystick)</td>
</tr>
<tr>
<td>getIdentification</td>
<td>(unsigned int joystick)</td>
</tr>
<tr>
<td>hasAxis</td>
<td>(unsigned int joystick, Axis axis)</td>
</tr>
<tr>
<td>isButtonPressed</td>
<td>(unsigned int joystick, unsigned int button)</td>
</tr>
<tr>
<td>isConnected</td>
<td>(unsigned int joystick)</td>
</tr>
<tr>
<td>PovX enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>PovY enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>R enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>U enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>update()</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>V enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>X enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>Y enum value</td>
<td>sf::Joystick</td>
</tr>
<tr>
<td>Z enum value</td>
<td>sf::Joystick</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Joystick::Identification Member List

This is the complete list of members for sf::Joystick::Identification, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification()</td>
<td>Defined in sf::Joystick::Identification</td>
</tr>
<tr>
<td>name</td>
<td>sf::Joystick::Identification</td>
</tr>
<tr>
<td>productId</td>
<td>sf::Joystick::Identification</td>
</tr>
<tr>
<td>vendorId</td>
<td>sf::Joystick::Identification</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
sf::Keyboard Member List

This is the complete list of members for sf::Keyboard, including all inheritance:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>enum value</td>
</tr>
<tr>
<td>Add</td>
<td>enum value</td>
</tr>
<tr>
<td>B</td>
<td>enum value</td>
</tr>
<tr>
<td>BackSlash</td>
<td>enum value</td>
</tr>
<tr>
<td>BackSpace</td>
<td>enum value</td>
</tr>
<tr>
<td>C</td>
<td>enum value</td>
</tr>
<tr>
<td>Comma</td>
<td>enum value</td>
</tr>
<tr>
<td>D</td>
<td>enum value</td>
</tr>
<tr>
<td>Dash</td>
<td>enum value</td>
</tr>
<tr>
<td>Delete</td>
<td>enum value</td>
</tr>
<tr>
<td>Divide</td>
<td>enum value</td>
</tr>
<tr>
<td>Down</td>
<td>enum value</td>
</tr>
<tr>
<td>E</td>
<td>enum value</td>
</tr>
<tr>
<td>End</td>
<td>enum value</td>
</tr>
<tr>
<td>Equal enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Escape enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F1 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F10 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F11 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F12 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F13 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F14 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F15 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F2 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F3 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F4 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F5 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F6 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F7 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F8 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>F9 enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>G enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>H enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>Home enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>I enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>Insert enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>isKeyPressed(Key key) static</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>J enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>K enum value</td>
<td>sf::Keyboard</td>
</tr>
<tr>
<td>Key</td>
<td>Enum Name</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>KeyCount</td>
<td>KeyCount</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>LAlt</td>
<td>LAlt</td>
</tr>
<tr>
<td>LBracket</td>
<td>LBracket</td>
</tr>
<tr>
<td>LControl</td>
<td>LControl</td>
</tr>
<tr>
<td>Left</td>
<td>Left</td>
</tr>
<tr>
<td>LShift</td>
<td>LShift</td>
</tr>
<tr>
<td>LSystem</td>
<td>LSystem</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Menu</td>
<td>Menu</td>
</tr>
<tr>
<td>Multiply</td>
<td>Multiply</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Num0</td>
<td>Num0</td>
</tr>
<tr>
<td>Num1</td>
<td>Num1</td>
</tr>
<tr>
<td>Num2</td>
<td>Num2</td>
</tr>
<tr>
<td>Num3</td>
<td>Num3</td>
</tr>
<tr>
<td>Num4</td>
<td>Num4</td>
</tr>
<tr>
<td>Num5</td>
<td>Num5</td>
</tr>
<tr>
<td>Num6</td>
<td>Num6</td>
</tr>
<tr>
<td>Num7</td>
<td>Num7</td>
</tr>
<tr>
<td>Num8</td>
<td>Num8</td>
</tr>
<tr>
<td>Num9</td>
<td>Num9</td>
</tr>
<tr>
<td>Numpad0</td>
<td>Numpad0</td>
</tr>
<tr>
<td>Numpad1</td>
<td>Numpad1</td>
</tr>
<tr>
<td>Key</td>
<td>Enum Value</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Numpad2</td>
<td>enum value</td>
</tr>
<tr>
<td>Numpad3</td>
<td>enum value</td>
</tr>
<tr>
<td>Numpad4</td>
<td>enum value</td>
</tr>
<tr>
<td>Numpad5</td>
<td>enum value</td>
</tr>
<tr>
<td>Numpad6</td>
<td>enum value</td>
</tr>
<tr>
<td>Numpad7</td>
<td>enum value</td>
</tr>
<tr>
<td>Numpad8</td>
<td>enum value</td>
</tr>
<tr>
<td>Numpad9</td>
<td>enum value</td>
</tr>
<tr>
<td>O</td>
<td>enum value</td>
</tr>
<tr>
<td>P</td>
<td>enum value</td>
</tr>
<tr>
<td>PageDown</td>
<td>enum value</td>
</tr>
<tr>
<td>PageUp</td>
<td>enum value</td>
</tr>
<tr>
<td>Pause</td>
<td>enum value</td>
</tr>
<tr>
<td>Period</td>
<td>enum value</td>
</tr>
<tr>
<td>Q</td>
<td>enum value</td>
</tr>
<tr>
<td>Quote</td>
<td>enum value</td>
</tr>
<tr>
<td>R</td>
<td>enum value</td>
</tr>
<tr>
<td>RAlt</td>
<td>enum value</td>
</tr>
<tr>
<td>RBracket</td>
<td>enum value</td>
</tr>
<tr>
<td>RControl</td>
<td>enum value</td>
</tr>
<tr>
<td>Return</td>
<td>enum value</td>
</tr>
<tr>
<td>Right</td>
<td>enum value</td>
</tr>
<tr>
<td>RShift</td>
<td>enum value</td>
</tr>
<tr>
<td>RSystem</td>
<td>enum value</td>
</tr>
<tr>
<td>S</td>
<td>enum value</td>
</tr>
<tr>
<td>SemiColon</td>
<td>enum value</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>setVirtualKeyboardVisible</td>
<td>(bool visible)</td>
</tr>
<tr>
<td>Slash</td>
<td>enum value</td>
</tr>
<tr>
<td>Space</td>
<td>enum value</td>
</tr>
<tr>
<td>Subtract</td>
<td>enum value</td>
</tr>
<tr>
<td>T</td>
<td>enum value</td>
</tr>
<tr>
<td>Tab</td>
<td>enum value</td>
</tr>
<tr>
<td>Tilde</td>
<td>enum value</td>
</tr>
<tr>
<td>U</td>
<td>enum value</td>
</tr>
<tr>
<td>Unknown</td>
<td>enum value</td>
</tr>
<tr>
<td>Up</td>
<td>enum value</td>
</tr>
<tr>
<td>V</td>
<td>enum value</td>
</tr>
<tr>
<td>W</td>
<td>enum value</td>
</tr>
<tr>
<td>X</td>
<td>enum value</td>
</tr>
<tr>
<td>Y</td>
<td>enum value</td>
</tr>
<tr>
<td>Z</td>
<td>enum value</td>
</tr>
</tbody>
</table>
sf::Mouse Member List

This is the complete list of members for `sf::Mouse`, including all inherited members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>enum name</td>
</tr>
<tr>
<td>ButtonCount</td>
<td>enum value</td>
</tr>
<tr>
<td>getPosition()</td>
<td></td>
</tr>
<tr>
<td>getPosition(const Window &amp;relativeTo)</td>
<td></td>
</tr>
<tr>
<td>HorizontalWheel</td>
<td>enum value</td>
</tr>
<tr>
<td>isButtonPressed(Button button)</td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>enum value</td>
</tr>
<tr>
<td>Middle</td>
<td>enum value</td>
</tr>
<tr>
<td>Right</td>
<td>enum value</td>
</tr>
<tr>
<td>setPosition(const Vector2i &amp;position)</td>
<td></td>
</tr>
<tr>
<td>setPosition(const Vector2i &amp;position, const Window &amp;relativeTo)</td>
<td></td>
</tr>
<tr>
<td>VerticalWheel</td>
<td>enum value</td>
</tr>
<tr>
<td>Wheel</td>
<td>enum name</td>
</tr>
<tr>
<td>XButton1</td>
<td>enum value</td>
</tr>
</tbody>
</table>
XButton2 enum value

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila  ::  Documentation generated by doxygen.
sf::Sensor Member List

This is the complete list of members for sf::Sensor, including all inherited members:

<table>
<thead>
<tr>
<th>Accelerometer enum value</th>
<th>sf::Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count enum value</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>getValue(Type sensor)</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>Gravity enum value</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>Gyroscope enum value</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>isAvailable(Type sensor)</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>Magnetometer enum value</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>Orientation enum value</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>setEnabled(Type sensor, bool enabled)</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>Type enum name</td>
<td>sf::Sensor</td>
</tr>
<tr>
<td>UserAcceleration enum value</td>
<td>sf::Sensor</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license. 
Copyright © Laurent Gomila :: Documentation generated by doxygen
sf::Touch Member List

This is the complete list of members for sf::Touch, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>sf::Touch</th>
<th>static</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPosition(unsigned int finger)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getPosition(unsigned int finger, const Window &amp;relativeTo)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isDown(unsigned int finger)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### sf::VideoMode Member List

This is the complete list of members for `sf::VideoMode`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitsPerPixel</td>
<td></td>
</tr>
<tr>
<td>getDesktopMode()</td>
<td></td>
</tr>
<tr>
<td>getFullScreenModes()</td>
<td></td>
</tr>
<tr>
<td>height</td>
<td></td>
</tr>
<tr>
<td>isValid() const</td>
<td></td>
</tr>
<tr>
<td>operator!= (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td></td>
</tr>
<tr>
<td>operator&lt; (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td></td>
</tr>
<tr>
<td>operator&lt;= (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td></td>
</tr>
<tr>
<td>operator== (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td></td>
</tr>
<tr>
<td>operator&gt; (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td></td>
</tr>
<tr>
<td>operator&gt;= (const VideoMode &amp;left, const VideoMode &amp;right)</td>
<td></td>
</tr>
<tr>
<td>VideoMode()</td>
<td></td>
</tr>
<tr>
<td>VideoMode(unsigned int modeWidth, unsigned int modeHeight, unsigned int width)</td>
<td></td>
</tr>
</tbody>
</table>
sf::Window Member List

This is the complete list of members for sf::Window, including all inherited members:

- close()
- create(VideoMode mode, const String &title, Uint32 style=Style::Default, const ContextSettings &settings=ContextSettings())
- create(WindowHandle handle, const ContextSettings &settings=ContextSettings())
- display()
- ensureGLContext()
- getPosition() const
- getSettings() const
- getSize() const
- getSystemHandle() const
- GlResource()
- hasFocus() const
- isOpen() const
- NonCopyable()
- onCreate()
onResize()
pollEvent(Event &event)
requestFocus()
setActive(bool active=true) const
setFramerateLimit(unsigned int limit)
setIcon(unsigned int width, unsigned int height, const Uint8 *pixels)
setJoystickThreshold(float threshold)
setKeyRepeatEnabled(bool enabled)
setMouseCursorGrabbed(bool grabbed)
setMouseCursorVisible(bool visible)
setPosition(const Vector2i &position)
setSize(const Vector2u &size)
setTitle(const String &title)
setVerticalSyncEnabled(bool enabled)
setVisible(bool visible)
waitEvent(Event &event)

Window()
Window(VideoMode mode, const String &title, Uint32 style=Style::Default)
Window(WindowHandle handle, const ContextSettings &settings=ContextSettings())
~GlResource()
~Window()
sf::Utf< 16 > Member List

This is the complete list of members for sf::Utf< 16 >, including all inherited members:

- **count**(In begin, In end)
- **decode**(In begin, In end, Uint32 &output, Uint32 replacement=0)
- **encode**(Uint32 input, Out output, Uint16 replacement=0)
- **fromAnsi**(In begin, In end, Out output, const std::locale &locale=std::locale())
- **fromLatin1**(In begin, In end, Out output)
- **fromWide**(In begin, In end, Out output)
- **next**(In begin, In end)
- **toAnsi**(In begin, In end, Out output, char replacement=0, const std::locale &locale=std::locale())
- **toLatin1**(In begin, In end, Out output, char replacement=0)
- **toUtf16**(In begin, In end, Out output)
- **toUtf32**(In begin, In end, Out output)
- **toUtf8**(In begin, In end, Out output)
- **toWide**(In begin, In end, Out output, wchar_t replacement=0)
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
sf::Utf< 32 > Member List

This is the complete list of members for sf::Utf< 32 >, including all inherited members.

- `count(In begin, In end)`
- `decode(In begin, In end, Uint32 &output, Uint32 replacement=0)`
- `decodeAnsi(In input, const std::locale &locale=std::locale())`
- `decodeWide(In input)`
- `encode(Uint32 input, Out output, Uint32 replacement=0)`
- `encodeAnsi(Uint32 codepoint, Out output, char replacement=0, const std::locale &locale=std::locale())`
- `encodeWide(Uint32 codepoint, Out output, wchar_t replacement=0)`
- `fromAnsi(In begin, In end, Out output, const std::locale &locale=std::locale())`
- `fromLatin1(In begin, In end, Out output)`
- `fromWide(In begin, In end, Out output)`
- `next(In begin, In end)`
- `toAnsi(In begin, In end, Out output, char replacement=0, const std::locale &locale=std::locale())`
- `toLatin1(In begin, In end, Out output, char replacement=0)`
- `toUtf16(In begin, In end, Out output)`
toUtf32(In begin, In end, Out output)
toUtf8(In begin, In end, Out output)
toWide(In begin, In end, Out output, wchar_t replacement=0)
sf::Utf< 8 > Member List

This is the complete list of members for sf::Utf< 8 >, including all inherited

- `count(In begin, In end)`
- `decode(In begin, In end, Uint32 &output, Uint32 replacement=0)`
- `encode(Uint32 input, Out output, Uint8 replacement=0)`
- `fromAnsi(In begin, In end, Out output, const std::locale &locale=std::locale())`
- `fromLatin1(In begin, In end, Out output)`
- `fromWide(In begin, In end, Out output)`
- `next(In begin, In end)`
- `toAnsi(In begin, In end, Out output, char replacement=0, const std::locale &locale)`
- `toLatin1(In begin, In end, Out output, char replacement=0)`
- `toUtf16(In begin, In end, Out output)`
- `toUtf32(In begin, In end, Out output)`
- `toUtf8(In begin, In end, Out output)`
- `toWide(In begin, In end, Out output, wchar_t replacement=0)`
SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
include Directory Reference
## Directories

| directory | SFML |

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.
SFML Directory Reference
Directories

<table>
<thead>
<tr>
<th>directory</th>
<th>Audio</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Graphics</td>
</tr>
<tr>
<td>directory</td>
<td>Network</td>
</tr>
<tr>
<td>directory</td>
<td>System</td>
</tr>
<tr>
<td>directory</td>
<td>Window</td>
</tr>
<tr>
<td>file</td>
<td>Audio.hpp [code]</td>
</tr>
<tr>
<td>--------</td>
<td>------------------</td>
</tr>
<tr>
<td>file</td>
<td>Config.hpp [code]</td>
</tr>
<tr>
<td>file</td>
<td>Graphics.hpp [code]</td>
</tr>
<tr>
<td>file</td>
<td>Main.hpp [code]</td>
</tr>
<tr>
<td>file</td>
<td>Network.hpp [code]</td>
</tr>
<tr>
<td>file</td>
<td>OpenGL.hpp [code]</td>
</tr>
<tr>
<td>file</td>
<td>System.hpp [code]</td>
</tr>
<tr>
<td>file</td>
<td>Window.hpp [code]</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license. Copyright © Laurent Gomila. :: Documentation generated by doxygen.
Audio Directory Reference
### Files

<table>
<thead>
<tr>
<th>File</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlResource.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Audio/Export.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>InputSoundFile.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Listener.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Music.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>OutputSoundFile.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Sound.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundBuffer.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundBufferRecorder.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundFileFactory.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundFileReader.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundFileWriter.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundRecorder.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundSource.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SoundStream.hpp</td>
<td>[code]</td>
</tr>
</tbody>
</table>
Graphics Directory Reference
<table>
<thead>
<tr>
<th>File</th>
<th>Name</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>file BlendMode.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file CircleShape.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Color.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file ConvexShape.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Drawable.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Graphics/Export.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Font.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Glsl.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Glyph.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Image.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file PrimitiveType.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file Rect.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file RectangleShape.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file RenderStates.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file RenderTarget.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>file RenderTexture.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>file</td>
<td>RenderWindow.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Shader.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Shape.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Sprite.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Text.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Texture.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Transform.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Transformable.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Vertex.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>VertexArray.hpp [code]</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>View.hpp [code]</td>
<td></td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila :: Documentation generated by doxygen.
System Directory Reference
### Files

<table>
<thead>
<tr>
<th>file</th>
<th>name</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td><strong>Clock.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>Err.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>System/Export.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>FileInputStream.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>InputStream.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>Lock.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>MemoryInputStream.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>Mutex.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>NativeActivity.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>NonCopyable.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>Sleep.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>String.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>Thread.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>ThreadLocal.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>ThreadLocalPtr.hpp</strong></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td><strong>Time.hpp</strong></td>
<td></td>
</tr>
</tbody>
</table>
file  Utf.hpp [code]

file  Vector2.hpp [code]

file  Vector3.hpp [code]

SFML is licensed under the terms and conditions of the zlib/png license.
Copyright © Laurent Gomila  :: Documentation generated by doxygen
Window Directory Reference
SFML is licensed under the terms and conditions of the zlib/png license.
Network Directory Reference
### Files

<table>
<thead>
<tr>
<th>File</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network/Export.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Ftp.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Http.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>IpAddress.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Packet.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>Socket.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SocketHandle.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>SocketSelector.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>TcpListener.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>TcpSocket.hpp</td>
<td>[code]</td>
</tr>
<tr>
<td>UdpSocket.hpp</td>
<td>[code]</td>
</tr>
</tbody>
</table>

SFML is licensed under the terms and conditions of the [zlib/png](https://zlib.net/) license.

Copyright © Laurent Gomila. :: Documentation generated by doxygen.
## Files

<table>
<thead>
<tr>
<th>File</th>
<th>mainpage.hpp</th>
</tr>
</thead>
</table>

SFML is licensed under the terms and conditions of the zlib/png license.

Copyright © Laurent Gomila :: Documentation generated by doxygen.