SFML

* [Main Page](http://docs.google.com/index.htm)
* [Modules](http://docs.google.com/modules.htm)
* [Classes](http://docs.google.com/annotated.htm)
* [Files](http://docs.google.com/files.htm)
* [Class List](http://docs.google.com/annotated.htm)
* [Class Index](http://docs.google.com/classes.htm)
* [Class Hierarchy](http://docs.google.com/hierarchy.htm)
* [Class Members](http://docs.google.com/functions.htm)
* **sf**
* [RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm)

[Public Member Functions](#_gjdgxs) | [Protected Member Functions](#_30j0zll) | [List of all members](http://docs.google.com/classsf_1_1RenderWindow-members.htm)

sf::RenderWindow Class Reference

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

[Window](http://docs.google.com/classsf_1_1Window.htm) that can serve as a target for 2D drawing. [More...](http://docs.google.com/classsf_1_1RenderWindow.htm#details)

#include <[RenderWindow.hpp](http://docs.google.com/RenderWindow_8hpp_source.htm)>

Inheritance diagram for sf::RenderWindow:



| Public Member Functions | |
| --- | --- |
|  | [RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm#a839bbf336bdcafb084dafc3076fc9021) () |
|  | Default constructor. |
|  | |
|  | [RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm#aebef983e01f677bf5a66cefc4d547647) ([VideoMode](http://docs.google.com/classsf_1_1VideoMode.htm) mode, const [String](http://docs.google.com/classsf_1_1String.htm) &title, Uint32 style=Style::Default, const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) &settings=[ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)()) |
|  | Construct a new window. |
|  | |
|  | [RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm#a25c0af7d515e710b6eebc9c6be952aa5) (WindowHandle handle, const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) &settings=[ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)()) |
|  | Construct the window from an existing control. |
|  | |
| virtual | [~RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm#a3407e36bfc1752d723140438a825365c) () |
|  | Destructor. |
|  | |
| virtual [Vector2u](http://docs.google.com/classsf_1_1Vector2.htm) | [getSize](http://docs.google.com/classsf_1_1RenderWindow.htm#a2c7ff414be32621a453745cf2a0f8a3e) () const |
|  | Get the size of the rendering region of the window. |
|  | |
| [Image](http://docs.google.com/classsf_1_1Image.htm) | [capture](http://docs.google.com/classsf_1_1RenderWindow.htm#a9bd8655d0bac83145bfc329ea7a6d538) () const |
|  | Copy the current contents of the window to an image. |
|  | |
| void | [create](http://docs.google.com/classsf_1_1Window.htm#a30e6edf2162f8dbff61023b9de5d961d) ([VideoMode](http://docs.google.com/classsf_1_1VideoMode.htm) mode, const [String](http://docs.google.com/classsf_1_1String.htm) &title, Uint32 style=Style::Default, const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) &settings=[ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)()) |
|  | Create (or recreate) the window. |
|  | |
| void | [create](http://docs.google.com/classsf_1_1Window.htm#acf67483dc21f08d65c8835b3889b41b2) (WindowHandle handle, const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) &settings=[ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)()) |
|  | Create (or recreate) the window from an existing control. |
|  | |
| void | [close](http://docs.google.com/classsf_1_1Window.htm#a99d1e030387b0c26f5995670504fe7b5) () |
|  | Close the window and destroy all the attached resources. |
|  | |
| bool | [isOpen](http://docs.google.com/classsf_1_1Window.htm#a5aa9c2b2b0e51d3423c2b66c80253337) () const |
|  | Tell whether or not the window is open. |
|  | |
| const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) & | [getSettings](http://docs.google.com/classsf_1_1Window.htm#a5a9d5c15facf25ad4d9b2b30caa0a2db) () const |
|  | Get the settings of the OpenGL context of the window. |
|  | |
| bool | [pollEvent](http://docs.google.com/classsf_1_1Window.htm#a338e996585faf82e93069858e3b531b7) ([Event](http://docs.google.com/classsf_1_1Event.htm) &event) |
|  | Pop the event on top of the event queue, if any, and return it. |
|  | |
| bool | [waitEvent](http://docs.google.com/classsf_1_1Window.htm#aaf02ab64fbc1d374eef3696df54137bc) ([Event](http://docs.google.com/classsf_1_1Event.htm) &event) |
|  | Wait for an event and return it. |
|  | |
| [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) | [getPosition](http://docs.google.com/classsf_1_1Window.htm#a2e6bc12612ea289afea8268fe37c8678) () const |
|  | Get the position of the window. |
|  | |
| void | [setPosition](http://docs.google.com/classsf_1_1Window.htm#a6c4078bfbf61c29bfc4b4732ce764f17) (const [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) &position) |
|  | Change the position of the window on screen. |
|  | |
| void | [setSize](http://docs.google.com/classsf_1_1Window.htm#ad6513418bb2963347cd1819a1810524d) (const [Vector2u](http://docs.google.com/classsf_1_1Vector2.htm) size) |
|  | Change the size of the rendering region of the window. |
|  | |
| void | [setTitle](http://docs.google.com/classsf_1_1Window.htm#a3b3f3513bb6be90f5cd456c20b5fd5fa) (const [String](http://docs.google.com/classsf_1_1String.htm) &title) |
|  | Change the title of the window. |
|  | |
| void | [setIcon](http://docs.google.com/classsf_1_1Window.htm#a63af61e026fba08e3153fd013620bcc0) (unsigned int width, unsigned int height, const Uint8 \*pixels) |
|  | Change the window's icon. |
|  | |
| void | [setVisible](http://docs.google.com/classsf_1_1Window.htm#a160f7f11a207603d7e99ce606e749703) (bool visible) |
|  | Show or hide the window. |
|  | |
| void | [setVerticalSyncEnabled](http://docs.google.com/classsf_1_1Window.htm#a59041c4556e0351048f8aff366034f61) (bool enabled) |
|  | Enable or disable vertical synchronization. |
|  | |
| void | [setMouseCursorVisible](http://docs.google.com/classsf_1_1Window.htm#aad3991c25e0a83afbb4d62febf9b7b14) (bool visible) |
|  | Show or hide the mouse cursor. |
|  | |
| void | [setKeyRepeatEnabled](http://docs.google.com/classsf_1_1Window.htm#aef9f2b14c10ecba8a8df95dd51c5bb73) (bool enabled) |
|  | Enable or disable automatic key-repeat. |
|  | |
| void | [setFramerateLimit](http://docs.google.com/classsf_1_1Window.htm#af4322d315baf93405bf0d5087ad5e784) (unsigned int limit) |
|  | Limit the framerate to a maximum fixed frequency. |
|  | |
| void | [setJoystickThreshold](http://docs.google.com/classsf_1_1Window.htm#aa45b8f54e29a6f59f1fc7ee66b2fab68) (float threshold) |
|  | Change the joystick threshold. |
|  | |
| bool | [setActive](http://docs.google.com/classsf_1_1Window.htm#a17ccf8ece0ce0bf2f1e6698bcfa29731) (bool active=true) const |
|  | Activate or deactivate the window as the current target for OpenGL rendering. |
|  | |
| void | [display](http://docs.google.com/classsf_1_1Window.htm#adabf839cb103ac96cfc82f781638772a) () |
|  | Display on screen what has been rendered to the window so far. |
|  | |
| WindowHandle | [getSystemHandle](http://docs.google.com/classsf_1_1Window.htm#a26368e7162229f8637c34d80ab0f138e) () const |
|  | Get the OS-specific handle of the window. |
|  | |
| void | [clear](http://docs.google.com/classsf_1_1RenderTarget.htm#a6bb6f0ba348f2b1e2f46114aeaf60f26) (const [Color](http://docs.google.com/classsf_1_1Color.htm) &color=[Color](http://docs.google.com/classsf_1_1Color.htm)(0, 0, 0, 255)) |
|  | Clear the entire target with a single color. |
|  | |
| void | [setView](http://docs.google.com/classsf_1_1RenderTarget.htm#a063db6dd0a14913504af30e50cb6d946) (const [View](http://docs.google.com/classsf_1_1View.htm) &view) |
|  | Change the current active view. |
|  | |
| const [View](http://docs.google.com/classsf_1_1View.htm) & | [getView](http://docs.google.com/classsf_1_1RenderTarget.htm#a98f721cc6dc11478922427fedfb2288b) () const |
|  | Get the view currently in use in the render target. |
|  | |
| const [View](http://docs.google.com/classsf_1_1View.htm) & | [getDefaultView](http://docs.google.com/classsf_1_1RenderTarget.htm#a718b1aa6296bf855171699cc18251ced) () const |
|  | Get the default view of the render target. |
|  | |
| [IntRect](http://docs.google.com/classsf_1_1Rect.htm) | [getViewport](http://docs.google.com/classsf_1_1RenderTarget.htm#aae035b0d45f87a0da2a28a0de6ba1086) (const [View](http://docs.google.com/classsf_1_1View.htm) &view) const |
|  | Get the viewport of a view, applied to this render target. |
|  | |
| [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) | [mapPixelToCoords](http://docs.google.com/classsf_1_1RenderTarget.htm#a2b0cab0e4c6af29d4efaba149d28116d) (const [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) &point) const |
|  | Convert a point from target coordinates to world coordinates, using the current view. |
|  | |
| [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) | [mapPixelToCoords](http://docs.google.com/classsf_1_1RenderTarget.htm#a46eb08f775dd1420d6207ea87dde6e54) (const [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) &point, const [View](http://docs.google.com/classsf_1_1View.htm) &view) const |
|  | Convert a point from target coordinates to world coordinates. |
|  | |
| [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) | [mapCoordsToPixel](http://docs.google.com/classsf_1_1RenderTarget.htm#aa0c11e1989573f2cce64c621205f8e83) (const [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) &point) const |
|  | Convert a point from world coordinates to target coordinates, using the current view. |
|  | |
| [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) | [mapCoordsToPixel](http://docs.google.com/classsf_1_1RenderTarget.htm#a7a2d427bdb9bd8f9f456fcf82813aa60) (const [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) &point, const [View](http://docs.google.com/classsf_1_1View.htm) &view) const |
|  | Convert a point from world coordinates to target coordinates. |
|  | |
| void | [draw](http://docs.google.com/classsf_1_1RenderTarget.htm#a12417a3bcc245c41d957b29583556f39) (const [Drawable](http://docs.google.com/classsf_1_1Drawable.htm) &drawable, const [RenderStates](http://docs.google.com/classsf_1_1RenderStates.htm) &states=[RenderStates::Default](http://docs.google.com/classsf_1_1RenderStates.htm#ad29672df29f19ce50c3021d95f2bb062)) |
|  | Draw a drawable object to the render-target. |
|  | |
| void | [draw](http://docs.google.com/classsf_1_1RenderTarget.htm#ab636d7363f6681077361ee274ba89a8d) (const [Vertex](http://docs.google.com/classsf_1_1Vertex.htm) \*vertices, unsigned int vertexCount, [PrimitiveType](http://docs.google.com/group__graphics.htm#ga5ee56ac1339984909610713096283b1b) type, const [RenderStates](http://docs.google.com/classsf_1_1RenderStates.htm) &states=[RenderStates::Default](http://docs.google.com/classsf_1_1RenderStates.htm#ad29672df29f19ce50c3021d95f2bb062)) |
|  | Draw primitives defined by an array of vertices. |
|  | |
| void | [pushGLStates](http://docs.google.com/classsf_1_1RenderTarget.htm#a8d1998464ccc54e789aaf990242b47f7) () |
|  | Save the current OpenGL render states and matrices. |
|  | |
| void | [popGLStates](http://docs.google.com/classsf_1_1RenderTarget.htm#ad5a98401113df931ddcd54c080f7aa8e) () |
|  | Restore the previously saved OpenGL render states and matrices. |
|  | |
| void | [resetGLStates](http://docs.google.com/classsf_1_1RenderTarget.htm#aac7504990d27dada4bfe3c7866920765) () |
|  | Reset the internal OpenGL states so that the target is ready for drawing. |
|  | |

| Protected Member Functions | |
| --- | --- |
| virtual void | [onCreate](http://docs.google.com/classsf_1_1RenderWindow.htm#a5bef0040b0fa87bed9fbd459c980d53a) () |
|  | Function called after the window has been created. |
|  | |
| virtual void | [onResize](http://docs.google.com/classsf_1_1RenderWindow.htm#a5c85fe482313562d33ffd24a194b6fef) () |
|  | Function called after the window has been resized. |
|  | |
| void | [initialize](http://docs.google.com/classsf_1_1RenderTarget.htm#af530274b34159d644e509b4b4dc43eb7) () |
|  | Performs the common initialization step after creation. |
|  | |

## Detailed Description

[Window](http://docs.google.com/classsf_1_1Window.htm) that can serve as a target for 2D drawing.

[sf::RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm) is the main class of the Graphics module.

It defines an OS window that can be painted using the other classes of the graphics module.

[sf::RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm) is derived from [sf::Window](http://docs.google.com/classsf_1_1Window.htm), thus it inherits all its features: events, window management, OpenGL rendering, etc. See the documentation of [sf::Window](http://docs.google.com/classsf_1_1Window.htm) for a more complete description of all these features, as well as code examples.

On top of that, [sf::RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm) adds more features related to 2D drawing with the graphics module (see its base class [sf::RenderTarget](http://docs.google.com/classsf_1_1RenderTarget.htm) for more details). Here is a typical rendering and event loop with a [sf::RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm):

// Declare and create a new render-window

[sf::RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm) window([sf::VideoMode](http://docs.google.com/classsf_1_1VideoMode.htm)(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](http://docs.google.com/classsf_1_1Event.htm) event;

while (window.pollEvent(event))

{

// Request for closing the window

if (event.[type](http://docs.google.com/classsf_1_1Event.htm#adf2f8044f713fd9d6019077b0d1ffe0a) == [sf::Event::Closed](http://docs.google.com/classsf_1_1Event.htm#af41fa9ed45c02449030699f671331d4aa316e4212e083f1dce79efd8d9e9c0a95))

window.close();

}

// Clear the whole window before rendering a new frame

window.clear();

// Draw some graphical entities

window.draw(sprite);

window.draw(circle);

window.draw(text);

// End the current frame and display its contents on screen

window.display();

}

Like [sf::Window](http://docs.google.com/classsf_1_1Window.htm), [sf::RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm) is still able to render direct OpenGL stuff. It is even possible to mix together OpenGL calls and regular SFML drawing commands.

// Create the render window

[sf::RenderWindow](http://docs.google.com/classsf_1_1RenderWindow.htm) window([sf::VideoMode](http://docs.google.com/classsf_1_1VideoMode.htm)(800, 600), "SFML OpenGL");

// Create a sprite and a text to display

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

[sf::Text](http://docs.google.com/classsf_1_1Text.htm) 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](http://docs.google.com/classsf_1_1Window.htm), [sf::RenderTarget](http://docs.google.com/classsf_1_1RenderTarget.htm), [sf::RenderTexture](http://docs.google.com/classsf_1_1RenderTexture.htm), [sf::View](http://docs.google.com/classsf_1_1View.htm)

Definition at line [44](http://docs.google.com/RenderWindow_8hpp_source.htm#l00044) of file [RenderWindow.hpp](http://docs.google.com/RenderWindow_8hpp_source.htm).

## Constructor & Destructor Documentation

| sf::RenderWindow::RenderWindow | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Default constructor.

This constructor doesn't actually create the window, use the other constructors or call "create" to do so.

| sf::RenderWindow::RenderWindow | ( | [VideoMode](http://docs.google.com/classsf_1_1VideoMode.htm) | *mode*, |
| --- | --- | --- | --- |
|  |  | const [String](http://docs.google.com/classsf_1_1String.htm) & | *title*, |
|  |  | Uint32 | *style* = Style::Default, |
|  |  | const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) & | *settings* = [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)() |
|  | ) |  |  |

Construct a new window.

This constructor creates the window with the size and pixel depth defined in *mode*. An optional style can be passed to customize the look and behaviour of the window (borders, title bar, resizable, closable, ...).

The fourth 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

| mode | Video mode to use (defines the width, height and depth of the rendering area of the window) |
| --- | --- |
| title | Title of the window |
| style | [Window](http://docs.google.com/classsf_1_1Window.htm) style |
| settings | Additional settings for the underlying OpenGL context |

| | sf::RenderWindow::RenderWindow | ( | WindowHandle | *handle*, | | --- | --- | --- | --- | |  |  | const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) & | *settings* = [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)() | |  | ) |  |  | | explicit |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Construct the window from an existing control.

Use this constructor if you want to create an SFML rendering area into an already existing control.

The fourth 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 |
| --- | --- |
| settings | Additional settings for the underlying OpenGL context |

| | virtual sf::RenderWindow::~RenderWindow | ( |  | ) |  | | --- | --- | --- | --- | --- | | virtual |
| --- | --- | --- | --- | --- | --- | --- |

Destructor.

Closes the window and free all the resources attached to it.

## Member Function Documentation

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

Copy the current contents of the window to an image.

This is a slow operation, whose main purpose is to make screenshots of the application. If you want to update an image with the contents of the window and then use it for drawing, you should rather use a [sf::Texture](http://docs.google.com/classsf_1_1Texture.htm) and its update(Window&) function. You can also draw things directly to a texture with the [sf::RenderTexture](http://docs.google.com/classsf_1_1RenderTexture.htm) class.

Returns[Image](http://docs.google.com/classsf_1_1Image.htm) containing the captured contents

| | void sf::RenderTarget::clear | ( | const [Color](http://docs.google.com/classsf_1_1Color.htm) & | *color* = [Color](http://docs.google.com/classsf_1_1Color.htm)(0, 0, 0, 255) | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Clear the entire target with a single color.

This function is usually called once every frame, to clear the previous contents of the target.

Parameters

| color | Fill color to use to clear the render target |
| --- | --- |

| | void sf::Window::close | ( |  | ) |  | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Close the window and destroy all the attached resources.

After calling this function, the [sf::Window](http://docs.google.com/classsf_1_1Window.htm) instance remains valid and you can call [create()](http://docs.google.com/classsf_1_1Window.htm#a30e6edf2162f8dbff61023b9de5d961d) to recreate the window. All other functions such as [pollEvent()](http://docs.google.com/classsf_1_1Window.htm#a338e996585faf82e93069858e3b531b7) or [display()](http://docs.google.com/classsf_1_1Window.htm#adabf839cb103ac96cfc82f781638772a) will still work (i.e. you don't have to test [isOpen()](http://docs.google.com/classsf_1_1Window.htm#a5aa9c2b2b0e51d3423c2b66c80253337) every time), and will have no effect on closed windows.

| | void sf::Window::create | ( | [VideoMode](http://docs.google.com/classsf_1_1VideoMode.htm) | *mode*, | | --- | --- | --- | --- | |  |  | const [String](http://docs.google.com/classsf_1_1String.htm) & | *title*, | |  |  | Uint32 | *style* = Style::Default, | |  |  | const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) & | *settings* = [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)() | |  | ) |  |  | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Create (or recreate) the window.

If the window was already created, it closes it first. If *style* contains Style::Fullscreen, then *mode* must be a valid video mode.

Parameters

| mode | Video mode to use (defines the width, height and depth of the rendering area of the window) |
| --- | --- |
| title | Title of the window |
| style | [Window](http://docs.google.com/classsf_1_1Window.htm) style |
| settings | Additional settings for the underlying OpenGL context |

| | void sf::Window::create | ( | WindowHandle | *handle*, | | --- | --- | --- | --- | |  |  | const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm) & | *settings* = [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)() | |  | ) |  |  | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

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.

Parameters

| handle | Platform-specific handle of the control |
| --- | --- |
| settings | Additional settings for the underlying OpenGL context |

| | void sf::Window::display | ( |  | ) |  | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Display on screen what has been rendered to the window so far.

This function is typically called after all OpenGL rendering has been done for the current frame, in order to show it on screen.

| | void sf::RenderTarget::draw | ( | const [Drawable](http://docs.google.com/classsf_1_1Drawable.htm) & | *drawable*, | | --- | --- | --- | --- | |  |  | const [RenderStates](http://docs.google.com/classsf_1_1RenderStates.htm) & | *states* = [RenderStates::Default](http://docs.google.com/classsf_1_1RenderStates.htm#ad29672df29f19ce50c3021d95f2bb062) | |  | ) |  |  | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

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](http://docs.google.com/classsf_1_1Vertex.htm) \* | *vertices*, | | --- | --- | --- | --- | |  |  | unsigned int | *vertexCount*, | |  |  | [PrimitiveType](http://docs.google.com/group__graphics.htm#ga5ee56ac1339984909610713096283b1b) | *type*, | |  |  | const [RenderStates](http://docs.google.com/classsf_1_1RenderStates.htm) & | *states* = [RenderStates::Default](http://docs.google.com/classsf_1_1RenderStates.htm#ad29672df29f19ce50c3021d95f2bb062) | |  | ) |  |  | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

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](http://docs.google.com/classsf_1_1View.htm)& sf::RenderTarget::getDefaultView | ( |  | ) | const | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

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.

ReturnsThe default view of the render target See Also[setView](http://docs.google.com/classsf_1_1RenderTarget.htm#a063db6dd0a14913504af30e50cb6d946), [getView](http://docs.google.com/classsf_1_1RenderTarget.htm#a98f721cc6dc11478922427fedfb2288b)

| | [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) sf::Window::getPosition | ( |  | ) | const | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Get the position of the window.

ReturnsPosition of the window, in pixels See Also[setPosition](http://docs.google.com/classsf_1_1Window.htm#a6c4078bfbf61c29bfc4b4732ce764f17)

| | const [ContextSettings](http://docs.google.com/structsf_1_1ContextSettings.htm)& sf::Window::getSettings | ( |  | ) | const | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Get the settings of the OpenGL context of the window.

Note that these settings may be different from what was passed to the constructor or the [create()](http://docs.google.com/classsf_1_1Window.htm#a30e6edf2162f8dbff61023b9de5d961d) function, if one or more settings were not supported. In this case, SFML chose the closest match.

ReturnsStructure containing the OpenGL context settings

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

Get the size of the rendering region of the window.

The size doesn't include the titlebar and borders of the window.

ReturnsSize in pixels

Implements [sf::RenderTarget](http://docs.google.com/classsf_1_1RenderTarget.htm#a2e5ade2457d9fb4c4907ae5b3d9e94a5).

| | WindowHandle sf::Window::getSystemHandle | ( |  | ) | const | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Get the OS-specific handle of the window.

The type of the returned handle is sf::WindowHandle, which is a typedef to the handle 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 is fixed.

ReturnsSystem handle of the window

| | const [View](http://docs.google.com/classsf_1_1View.htm)& sf::RenderTarget::getView | ( |  | ) | const | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Get the view currently in use in the render target.

ReturnsThe view object that is currently used See Also[setView](http://docs.google.com/classsf_1_1RenderTarget.htm#a063db6dd0a14913504af30e50cb6d946), [getDefaultView](http://docs.google.com/classsf_1_1RenderTarget.htm#a718b1aa6296bf855171699cc18251ced)

| | [IntRect](http://docs.google.com/classsf_1_1Rect.htm) sf::RenderTarget::getViewport | ( | const [View](http://docs.google.com/classsf_1_1View.htm) & | *view* | ) | const | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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 this ratio to the current 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 |
| --- | --- |

ReturnsViewport rectangle, expressed in pixels

| | void sf::RenderTarget::initialize | ( |  | ) |  | | --- | --- | --- | --- | --- | | protectedinherited |
| --- | --- | --- | --- | --- | --- | --- |

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::Window::isOpen | ( |  | ) | const | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Tell whether or not the window is open.

This function returns whether or not the window exists. Note that a hidden window (setVisible(false)) is open (therefore this function would return true).

ReturnsTrue if the window is open, false if it has been closed

| | [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) sf::RenderTarget::mapCoordsToPixel | ( | const [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) & | *point* | ) | const | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Convert a point from world coordinates to target coordinates, using the current view.

This function is an overload of the mapCoordsToPixel function that implicitely uses the current view. It is equivalent to:

target.[mapCoordsToPixel](http://docs.google.com/classsf_1_1RenderTarget.htm#aa0c11e1989573f2cce64c621205f8e83)(point, target.[getView](http://docs.google.com/classsf_1_1RenderTarget.htm#a98f721cc6dc11478922427fedfb2288b)());

Parameters

| point | Point to convert |
| --- | --- |

ReturnsThe converted point, in target coordinates (pixels) See Also[mapPixelToCoords](http://docs.google.com/classsf_1_1RenderTarget.htm#a2b0cab0e4c6af29d4efaba149d28116d)

| | [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) sf::RenderTarget::mapCoordsToPixel | ( | const [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) & | *point*, | | --- | --- | --- | --- | |  |  | const [View](http://docs.google.com/classsf_1_1View.htm) & | *view* | |  | ) |  | const | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Convert a point from world coordinates to target coordinates.

This function finds the pixel of the render-target that matches the given 2D point. In other words, it goes through the same process as the graphics card, to compute the final position of a rendered point.

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, ie. 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 of the function if you want to use the current view of the render-target.

Parameters

| point | Point to convert |
| --- | --- |
| view | The view to use for converting the point |

ReturnsThe converted point, in target coordinates (pixels) See Also[mapPixelToCoords](http://docs.google.com/classsf_1_1RenderTarget.htm#a2b0cab0e4c6af29d4efaba149d28116d)

| | [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) sf::RenderTarget::mapPixelToCoords | ( | const [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) & | *point* | ) | const | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Convert a point from target coordinates to world coordinates, using the current view.

This function is an overload of the mapPixelToCoords function that implicitely uses the current view. It is equivalent to:

target.[mapPixelToCoords](http://docs.google.com/classsf_1_1RenderTarget.htm#a2b0cab0e4c6af29d4efaba149d28116d)(point, target.[getView](http://docs.google.com/classsf_1_1RenderTarget.htm#a98f721cc6dc11478922427fedfb2288b)());

Parameters

| point | Pixel to convert |
| --- | --- |

ReturnsThe converted point, in "world" coordinates See Also[mapCoordsToPixel](http://docs.google.com/classsf_1_1RenderTarget.htm#aa0c11e1989573f2cce64c621205f8e83)

| | [Vector2f](http://docs.google.com/classsf_1_1Vector2.htm) sf::RenderTarget::mapPixelToCoords | ( | const [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) & | *point*, | | --- | --- | --- | --- | |  |  | const [View](http://docs.google.com/classsf_1_1View.htm) & | *view* | |  | ) |  | const | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Convert a point from target coordinates to world coordinates.

This function finds the 2D position that matches the given pixel of the render-target. In other words, 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, ie. 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 (or object) is located below the mouse cursor.

This version uses a custom view for calculations, see the other overload of the function 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 |

ReturnsThe converted point, in "world" units See Also[mapCoordsToPixel](http://docs.google.com/classsf_1_1RenderTarget.htm#aa0c11e1989573f2cce64c621205f8e83)

| | virtual void sf::RenderWindow::onCreate | ( |  | ) |  | | --- | --- | --- | --- | --- | | protectedvirtual |
| --- | --- | --- | --- | --- | --- | --- |

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](http://docs.google.com/classsf_1_1Window.htm#a106633b9be49b27f83d4712689b493eb).

| | virtual void sf::RenderWindow::onResize | ( |  | ) |  | | --- | --- | --- | --- | --- | | protectedvirtual |
| --- | --- | --- | --- | --- | --- | --- |

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](http://docs.google.com/classsf_1_1Window.htm#a10f567a387da7b49f417f73321fcf64d).

| | bool sf::Window::pollEvent | ( | [Event](http://docs.google.com/classsf_1_1Event.htm) & | *event* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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 *event* 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](http://docs.google.com/classsf_1_1Event.htm) event;

while (window.pollEvent(event))

{

// process event...

}

Parameters

| event | [Event](http://docs.google.com/classsf_1_1Event.htm) to be returned |
| --- | --- |

ReturnsTrue if an event was returned, or false if the event queue was empty See Also[waitEvent](http://docs.google.com/classsf_1_1Window.htm#aaf02ab64fbc1d374eef3696df54137bc)

| | void sf::RenderTarget::popGLStates | ( |  | ) |  | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

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](http://docs.google.com/classsf_1_1RenderTarget.htm#a8d1998464ccc54e789aaf990242b47f7)

| | void sf::RenderTarget::pushGLStates | ( |  | ) |  | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

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. Example:

// 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 and matrices, even the ones you don't care about. Therefore it should be used wisely. It is provided for convenience, but the best results will be achieved if you handle OpenGL states yourself (because you know which states have really changed, and need to be saved and restored). Take a look at the ResetGLStates function if you do so.

See Also[popGLStates](http://docs.google.com/classsf_1_1RenderTarget.htm#ad5a98401113df931ddcd54c080f7aa8e)

| | void sf::RenderTarget::resetGLStates | ( |  | ) |  | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

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 that subsequent [draw()](http://docs.google.com/classsf_1_1RenderTarget.htm#a12417a3bcc245c41d957b29583556f39) calls will work as expected.

Example:

// OpenGL code here...

glPushAttrib(...);

window.resetGLStates();

window.draw(...);

window.draw(...);

glPopAttrib(...);

// OpenGL code here...

| | bool sf::Window::setActive | ( | bool | *active* = true | ) | const | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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.

Parameters

| active | True to activate, false to deactivate |
| --- | --- |

ReturnsTrue if operation was successful, false otherwise

| | void sf::Window::setFramerateLimit | ( | unsigned int | *limit* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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()](http://docs.google.com/classsf_1_1Window.htm#adabf839cb103ac96cfc82f781638772a) to ensure that the current frame lasted long enough to match the framerate limit. SFML will try to match the given limit as much as 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 requesting 60).

Parameters

| limit | Framerate limit, in frames per seconds (use 0 to disable limit) |
| --- | --- |

| | void sf::Window::setIcon | ( | unsigned int | *width*, | | --- | --- | --- | --- | |  |  | unsigned int | *height*, | |  |  | const Uint8 \* | *pixels* | |  | ) |  |  | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

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 |

See Also[setTitle](http://docs.google.com/classsf_1_1Window.htm#a3b3f3513bb6be90f5cd456c20b5fd5fa)

| | void sf::Window::setJoystickThreshold | ( | float | *threshold* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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] |
| --- | --- |

| | void sf::Window::setKeyRepeatEnabled | ( | bool | *enabled* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Enable or disable automatic key-repeat.

If key repeat is enabled, you will receive repeated KeyPressed events while keeping a key pressed. 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 |
| --- | --- |

| | void sf::Window::setMouseCursorVisible | ( | bool | *visible* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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 |
| --- | --- |

| | void sf::Window::setPosition | ( | const [Vector2i](http://docs.google.com/classsf_1_1Vector2.htm) & | *position* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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](http://docs.google.com/classsf_1_1Window.htm#a2e6bc12612ea289afea8268fe37c8678)

| | void sf::Window::setSize | ( | const [Vector2u](http://docs.google.com/classsf_1_1Vector2.htm) | *size* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Change the size of the rendering region of the window.

Parameters

| size | New size, in pixels |
| --- | --- |

See Also[getSize](http://docs.google.com/classsf_1_1Window.htm#ad2b55a731ba1680fe67292991ef1610e)

| | void sf::Window::setTitle | ( | const [String](http://docs.google.com/classsf_1_1String.htm) & | *title* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Change the title of the window.

Parameters

| title | New title |
| --- | --- |

See Also[setIcon](http://docs.google.com/classsf_1_1Window.htm#a63af61e026fba08e3153fd013620bcc0)

| | void sf::Window::setVerticalSyncEnabled | ( | bool | *enabled* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Enable or disable vertical synchronization.

Activating vertical synchronization will limit the number of frames displayed to the refresh rate of the monitor. This can avoid some visual artifacts, and limit the framerate to a good value (but not constant 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](http://docs.google.com/classsf_1_1View.htm) & | *view* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Change the current active view.

The view is like a 2D camera, it controls which part of the 2D scene is visible, and how it is viewed 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 [getDefaultView()](http://docs.google.com/classsf_1_1RenderTarget.htm#a718b1aa6296bf855171699cc18251ced) to this function.

Parameters

| view | New view to use |
| --- | --- |

See Also[getView](http://docs.google.com/classsf_1_1RenderTarget.htm#a98f721cc6dc11478922427fedfb2288b), [getDefaultView](http://docs.google.com/classsf_1_1RenderTarget.htm#a718b1aa6296bf855171699cc18251ced)

| | void sf::Window::setVisible | ( | bool | *visible* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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](http://docs.google.com/classsf_1_1Event.htm) & | *event* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Wait for an event and return it.

This function is blocking: if there's no pending event then it will wait until an event is received. After this function returns (and no error occured), the *event* object is always valid and filled properly. This function is typically used when you have a thread that is dedicated to events handling: you want to make this thread sleep as long as no new event is received.

[sf::Event](http://docs.google.com/classsf_1_1Event.htm) event;

if (window.waitEvent(event))

{

// process event...

}

Parameters

| event | [Event](http://docs.google.com/classsf_1_1Event.htm) to be returned |
| --- | --- |

ReturnsFalse if any error occured See Also[pollEvent](http://docs.google.com/classsf_1_1Window.htm#a338e996585faf82e93069858e3b531b7)

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

* [RenderWindow.hpp](http://docs.google.com/RenderWindow_8hpp_source.htm)

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