Documentation of SFML 2.5.1

Public Member Functions | Protected Member Functions | List of all members

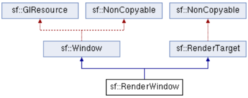
sf::RenderWindow Class Reference

Graphics module

Window that can serve as a target for 2D drawing. More...

`#include <RenderWindow.hpp>`

Inheritance diagram for sf::RenderWindow:



Public Member Functions

	RenderWindow () Default constructor. More...
	RenderWindow (VideoMode mode, const String &title, Uint32 style=Style::Default, const ContextSettings &settings=ContextSettings()) Construct a new window. More...
	RenderWindow (WindowHandle handle, const ContextSettings &settings=ContextSettings()) Construct the window from an existing control. More...
virtual	~RenderWindow () Destructor. More...
virtual Vector2u	getSize () const Get the size of the rendering region of the window. More...
bool	setActive (bool active=true) Activate or deactivate the window as the current target for OpenGL rendering. More...
Image	capture () const Copy the current contents of the window to an image. More...
void	create (VideoMode mode, const String &title, Uint32 style=Style::Default, const ContextSettings &settings=ContextSettings()) Create (or recreate) the window. More...
void	create (WindowHandle handle, const ContextSettings &settings=ContextSettings()) Create (or recreate) the window from an existing control. More...
void	close () Close the window and destroy all the attached resources. More...
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. More...
bool	pollEvent (Event &event) Pop the event on top of the event queue, if any, and return it. More...
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. More...
void	setSize (const Vector2u &size) Change the size of the rendering region of the window. More...
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...
void	setMouseCursorGrabbed (bool grabbed) Grab or release the mouse cursor. More...
void	setMouseCursor (const Cursor &cursor) Set the displayed cursor to a native system cursor. More...
void	setKeyRepeatEnabled (bool enabled) Enable or disable automatic key-repeat. More...
void	setFramerateLimit (unsigned int limit) Limit the framerate to a maximum fixed frequency. More...
void	setJoystickThreshold (float threshold) Change the joystick threshold. More...
bool	setActive (bool active=true) const Activate or deactivate the window as the current target for OpenGL rendering. More...
void	requestFocus () Request the current window to be made the active foreground window. More...
bool	hasFocus () const Check whether the window has the input focus. More...
void	display () Display on screen what has been rendered to the window so far. More...
WindowHandle	getSystemHandle () const Get the OS-specific handle of the window. More...
void	clear (const Color &color=Color(0, 0, 0, 255)) Clear the entire target with a single color. More...
void	setView (const View &view) Change the current active view. More...
const View &	getView () const Get the view currently in use in the render target. More...
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. More...
Vector2f	mapPixelToCoords (const Vector2i &point) const Convert a point from target coordinates to world coordinates, using the current view. More...
Vector2f	mapPixelToCoords (const Vector2i &point, const View &view) const Convert a point from target coordinates to world coordinates. More...
Vector2i	mapCoordsToPixel (const Vector2f &point) const Convert a point from world coordinates to target coordinates, using the current view. More...
Vector2i	mapCoordsToPixel (const Vector2f &point, const View &view) const Convert a point from world coordinates to target coordinates. More...
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, PrimitiveType type, const RenderStates &states=RenderStates::Default) Draw primitives defined by an array of vertices. More...
void	draw (const VertexBuffer &vertexBuffer, const RenderStates &states=RenderStates::Default) Draw primitives defined by a vertex buffer. More...
void	draw (const VertexBuffer &vertexBuffer, std::size_t firstVertex, std::size_t vertexCount, const RenderStates &states=RenderStates::Default) Draw primitives defined by a vertex buffer. More...
void	pushGLStates () Save the current OpenGL render states and matrices. More...
void	popGLStates () Restore the previously saved OpenGL render states and matrices. More...
void	resetGLStates () Reset the internal OpenGL states so that the target is ready for drawing. More...

Protected Member Functions

virtual void	onCreate () Function called after the window has been created. More...
virtual void	onResize () Function called after the window has been resized. More...
void	initialize ()

Performs the common initialization step after creation. More...

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 graphics module.

sf::RenderWindow is derived from sf::Window, thus it inherits all its features: events, window management, OpenGL rendering, etc. See the documentation of sf::Window for a more complete description of all these features, as well as code examples.

On top of that, sf::RenderWindow adds more features related to 2D drawing with the graphics module (see its base class sf::RenderTarget for more details). Here is a typical rendering and event loop with a sf::RenderWindow:

```
// 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);
   window.draw(text);

   // End the current frame and display its contents on screen
   window.display();
}
```

Like sf::Window, sf::RenderWindow 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 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 constructors or call create() to do so.

sf::RenderWindow::RenderWindow (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 in *mode*. An optional style can be passed to customize the look and behavior 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 style, a bitwise OR combination of sf::Style enumerators
settings	Additional settings for the underlying OpenGL context

sf::RenderWindow::RenderWindow (WindowHandle handle,
const ContextSettings & settings = `ContextSettings()`
)

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 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, *Window* on Linux/FreeBSD, *NSWindow* on OS X)
settings	Additional settings for the underlying OpenGL context

virtual sf::RenderWindow::~RenderWindow ( )

virtual

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 and copy its contents into an sf::Image instead.

```
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 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 and its update(Window&) function. You can also draw things directly to a texture with the sf::RenderTexture class.

Returns

Image containing the captured contents

```
void sf::RenderTarget::clear ( const Color & color = Color(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 instance remains valid and you can call create() to recreate the window. 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()
                        )
```

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.

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 the rendering area of the 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()
                        )
```

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.

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 ( )
```

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 &       drawable,
                              const RenderStates &   states = RenderStates::Default
                            )
```

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 *         vertices,
                              std::size_t            vertexCount,
                              PrimitiveType          type,
                              const RenderStates &   states = RenderStates::Default
                            )
```

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

```
void sf::RenderTarget::draw ( const VertexBuffer &   vertexBuffer,
                              const RenderStates &   states = RenderStates::Default
                            )
```

inherited

Draw primitives defined by a vertex buffer.

Parameters

vertexBuffer	Vertex buffer
states	Render states to use for drawing

```
void sf::RenderTarget::draw ( const VertexBuffer &   vertexBuffer,
                              std::size_t            firstVertex,
                              std::size_t            vertexCount,
                              const RenderStates &   states = RenderStates::Default
                            )
```

inherited

Draw primitives defined by a vertex buffer.

Parameters

vertexBuffer	Vertex buffer
firstVertex	Index of the first vertex to render
vertexCount	Number of vertices to render
states	Render states to use for drawing

```
const View& 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.

Returns

The default view of the render target

See also

setView, getView

```
Vector2i sf::Window::getPosition ( ) const
```

inherited

Get the position of the window.

Returns

Position of the window, in pixels

See also

setPosition

const ContextSettings& 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() function, if one or more settings were not supported. In this case, SFML chose the closest match.

Returns

Structure containing the OpenGL context settings

virtual Vector2u 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.

Returns

Size in pixels

Implements sf::RenderTarget.

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.

Returns

System handle of the window

const View& sf::RenderTarget::getView ( ) const

inherited

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

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

Returns

Viewport rectangle, expressed in pixels

bool sf::Window::hasFocus ( ) const

inherited

Check whether the window has the input focus.

At any given time, only one window may have the input focus to receive input events such as keystrokes or most mouse events.

Returns

True if window has focus, false otherwise

See also

requestFocus

void sf::RenderTarget::initialize ( )

protected inherited

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).

Returns

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

Vector2i sf::RenderTarget::mapCoordsToPixel (const Vector2f & 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 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) 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, i.e. 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

Returns

The converted point, in target coordinates (pixels)

See also

mapPixelToCoords

Vector2f sf::RenderTarget::mapPixelToCoords (const Vector2i & 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 implicitly uses the current view. It is 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

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, i.e. 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

Returns

The converted point, in "world" units

See also

mapCoordsToPixel

virtual void sf::RenderWindow::onCreate ( )

protected virtual

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 ( )

protected virtual

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)

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 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::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

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

void sf::Window::requestFocus ( )

inherited

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 input events such as keystrokes 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 be confused with setActive().

See also

hasFocus

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() calls will work as expected.

Example:

```
// OpenGL code here...
glPushAttrib(...);
window.resetGLStates();
window.draw(...);
window.draw(...);
glPopAttrib(...);
// OpenGL code here...
```

bool sf::RenderWindow::setActive (bool active = true)

virtual

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. This is not to be confused with requestFocus().

Parameters

active True to activate, false to deactivate

Returns

True if operation was successful, false otherwise

Reimplemented from sf::RenderTarget.

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. This is not to be confused 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)

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() 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. The pixels are copied, so you need not keep the source alive after calling this function.

See also

setTitle

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::setMouseCursor (const Cursor & cursor) inherited

Set the displayed cursor to a native system cursor.

Upon window creation, the arrow cursor is used by default.

Warning

The cursor must not be destroyed while in use by the window.

Features related to Cursor are not supported on iOS and Android.

Parameters

cursor	Native system cursor type to display

See also

sf::Cursor::loadFromSystem

sf::Cursor::loadFromPixels

void sf::Window::setMouseCursorGrabbed (bool grabbed) inherited

Grab or release the mouse cursor.

If set, grabs the mouse cursor inside this window's client area so it may no longer be moved outside its bounds. Note that grabbing is only active while the window has focus.

Parameters

grabbed	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 & **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

void sf::Window::setSize (const Vector2u & **size**) inherited

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**) inherited

Change the title of the window.

Parameters

title New title

See also

setIcon

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 & **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() to this function.

Parameters

view New view to use

See also

getView, getDefaultView

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 & 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 occurred), 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 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