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**
* [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm)

[Public Member Functions](#_gjdgxs) | [List of all members](http://docs.google.com/classsf_1_1SocketSelector-members.htm)

sf::SocketSelector Class Reference

[Network module](http://docs.google.com/group__network.htm)

Multiplexer that allows to read from multiple sockets. [More...](http://docs.google.com/classsf_1_1SocketSelector.htm#details)

#include <[SocketSelector.hpp](http://docs.google.com/SocketSelector_8hpp_source.htm)>

| Public Member Functions | |
| --- | --- |
|  | [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm#a741959c5158aeb1e4457cad47d90f76b) () |
|  | Default constructor. |
|  | |
|  | [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm#a50b1b955eb7ecb2e7c2764f3f4722fbf) (const [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm) &copy) |
|  | Copy constructor. |
|  | |
|  | [~SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm#a9069cd61208260b8ed9cf233afa1f73d) () |
|  | Destructor. |
|  | |
| void | [add](http://docs.google.com/classsf_1_1SocketSelector.htm#ade952013232802ff7b9b33668f8d2096) ([Socket](http://docs.google.com/classsf_1_1Socket.htm) &socket) |
|  | Add a new socket to the selector. |
|  | |
| void | [remove](http://docs.google.com/classsf_1_1SocketSelector.htm#a98b6ab693a65b82caa375639232357c1) ([Socket](http://docs.google.com/classsf_1_1Socket.htm) &socket) |
|  | Remove a socket from the selector. |
|  | |
| void | [clear](http://docs.google.com/classsf_1_1SocketSelector.htm#a76e650acb0199d4be91e90a493fbc91a) () |
|  | Remove all the sockets stored in the selector. |
|  | |
| bool | [wait](http://docs.google.com/classsf_1_1SocketSelector.htm#a9cfda5475f17925e65889394d70af702) ([Time](http://docs.google.com/classsf_1_1Time.htm) timeout=[Time::Zero](http://docs.google.com/classsf_1_1Time.htm#a8db127b632fa8da21550e7282af11fa0)) |
|  | Wait until one or more sockets are ready to receive. |
|  | |
| bool | [isReady](http://docs.google.com/classsf_1_1SocketSelector.htm#a8e67b463db05eadb4d356992c896833c) ([Socket](http://docs.google.com/classsf_1_1Socket.htm) &socket) const |
|  | Test a socket to know if it is ready to receive data. |
|  | |
| [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm) & | [operator=](http://docs.google.com/classsf_1_1SocketSelector.htm#ae6395c7a8d29a9ea14939cc5d1ba3a33) (const [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm) &right) |
|  | Overload of assignment operator. |
|  | |

## Detailed Description

Multiplexer that allows to read from multiple sockets.

[Socket](http://docs.google.com/classsf_1_1Socket.htm) 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; with selectors, a single thread can handle all the sockets.

All types of sockets can be used in a selector:

* [sf::TcpListener](http://docs.google.com/classsf_1_1TcpListener.htm)
* [sf::TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm)
* [sf::UdpSocket](http://docs.google.com/classsf_1_1UdpSocket.htm)

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 oustide 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:

// Create a socket to listen to new connections

[sf::TcpListener](http://docs.google.com/classsf_1_1TcpListener.htm) listener;

listener.[listen](http://docs.google.com/classsf_1_1TcpListener.htm#a409d9350d3abfea9636df8cf4a61004e)(55001);

// Create a list to store the future clients

std::list<sf::TcpSocket\*> clients;

// Create a selector

[sf::SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm) selector;

// Add the listener to the selector

selector.[add](http://docs.google.com/classsf_1_1SocketSelector.htm#ade952013232802ff7b9b33668f8d2096)(listener);

// Endless loop that waits for new connections

while (running)

{

// Make the selector wait for data on any socket

if (selector.[wait](http://docs.google.com/classsf_1_1SocketSelector.htm#a9cfda5475f17925e65889394d70af702)())

{

// Test the listener

if (selector.[isReady](http://docs.google.com/classsf_1_1SocketSelector.htm#a8e67b463db05eadb4d356992c896833c)(listener))

{

// The listener is ready: there is a pending connection

[sf::TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm)\* client = new [sf::TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm);

if (listener.[accept](http://docs.google.com/classsf_1_1TcpListener.htm#ae2c83ce5a64d50b68180c46bef0a7346)(\*client) == [sf::Socket::Done](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dca1de3a85bc56d3ae85b3d0f3cfd04ae90))

{

// 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](http://docs.google.com/classsf_1_1SocketSelector.htm#ade952013232802ff7b9b33668f8d2096)(\*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](http://docs.google.com/classsf_1_1TcpSocket.htm)& client = \*\*it;

if (selector.[isReady](http://docs.google.com/classsf_1_1SocketSelector.htm#a8e67b463db05eadb4d356992c896833c)(client))

{

// The client has sent some data, we can receive it

[sf::Packet](http://docs.google.com/classsf_1_1Packet.htm) packet;

if (client.[receive](http://docs.google.com/classsf_1_1TcpSocket.htm#a90ce50811ea61d4f00efc62bb99ae1af)(packet) == [sf::Socket::Done](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dca1de3a85bc56d3ae85b3d0f3cfd04ae90))

{

...

}

}

}

}

}

}

See Also[sf::Socket](http://docs.google.com/classsf_1_1Socket.htm)

Definition at line [43](http://docs.google.com/SocketSelector_8hpp_source.htm#l00043) of file [SocketSelector.hpp](http://docs.google.com/SocketSelector_8hpp_source.htm).

## Constructor & Destructor Documentation

| sf::SocketSelector::SocketSelector | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Default constructor.

| sf::SocketSelector::SocketSelector | ( | const [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm) & | *copy* | ) |  |
| --- | --- | --- | --- | --- | --- |

Copy constructor.

Parameters

| copy | Instance to copy |
| --- | --- |

| sf::SocketSelector::~SocketSelector | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Destructor.

## Member Function Documentation

| void sf::SocketSelector::add | ( | [Socket](http://docs.google.com/classsf_1_1Socket.htm) & | *socket* | ) |  |
| --- | --- | --- | --- | --- | --- |

Add a new socket to the selector.

This function keeps a weak reference to the socket, so you have to make sure that the socket 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](http://docs.google.com/classsf_1_1SocketSelector.htm#a98b6ab693a65b82caa375639232357c1), [clear](http://docs.google.com/classsf_1_1SocketSelector.htm#a76e650acb0199d4be91e90a493fbc91a)

| 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 that the selector has to external sockets.

See Also[add](http://docs.google.com/classsf_1_1SocketSelector.htm#ade952013232802ff7b9b33668f8d2096), [remove](http://docs.google.com/classsf_1_1SocketSelector.htm#a98b6ab693a65b82caa375639232357c1)

| bool sf::SocketSelector::isReady | ( | [Socket](http://docs.google.com/classsf_1_1Socket.htm) & | *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 to receive data. 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](http://docs.google.com/classsf_1_1TcpListener.htm), this means that it is ready to accept a new connection.

Parameters

| socket | [Socket](http://docs.google.com/classsf_1_1Socket.htm) to test |
| --- | --- |

ReturnsTrue if the socket is ready to read, false otherwise See Also[isReady](http://docs.google.com/classsf_1_1SocketSelector.htm#a8e67b463db05eadb4d356992c896833c)

| [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm)& sf::SocketSelector::operator= | ( | const [SocketSelector](http://docs.google.com/classsf_1_1SocketSelector.htm) & | *right* | ) |  |
| --- | --- | --- | --- | --- | --- |

Overload of assignment operator.

Parameters

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

ReturnsReference to self

| void sf::SocketSelector::remove | ( | [Socket](http://docs.google.com/classsf_1_1Socket.htm) & | *socket* | ) |  |
| --- | --- | --- | --- | --- | --- |

Remove a socket from the selector.

This function doesn't destroy the socket, it simply removes the reference that the selector has to it.

Parameters

| socket | Reference to the socket to remove |
| --- | --- |

See Also[add](http://docs.google.com/classsf_1_1SocketSelector.htm#ade952013232802ff7b9b33668f8d2096), [clear](http://docs.google.com/classsf_1_1SocketSelector.htm#a76e650acb0199d4be91e90a493fbc91a)

| bool sf::SocketSelector::wait | ( | [Time](http://docs.google.com/classsf_1_1Time.htm) | *timeout* = [Time::Zero](http://docs.google.com/classsf_1_1Time.htm#a8db127b632fa8da21550e7282af11fa0) | ) |  |
| --- | --- | --- | --- | --- | --- |

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 be received. To know which sockets are ready, use the isReady function. If you use a timeout and no socket is ready before the timeout is over, the function returns false.

Parameters

| timeout | Maximum time to wait, (use [Time::Zero](http://docs.google.com/classsf_1_1Time.htm#a8db127b632fa8da21550e7282af11fa0) for infinity) |
| --- | --- |

ReturnsTrue if there are sockets ready, false otherwise See Also[isReady](http://docs.google.com/classsf_1_1SocketSelector.htm#a8e67b463db05eadb4d356992c896833c)

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

* [SocketSelector.hpp](http://docs.google.com/SocketSelector_8hpp_source.htm)

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