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**
* [TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm)

[Classes](#3dy6vkm) | [Public Types](#_gjdgxs) | [Public Member Functions](#_30j0zll) | [Protected Types](#_1fob9te) | [Protected Member Functions](#_3znysh7) | [Friends](#_2et92p0) | [List of all members](http://docs.google.com/classsf_1_1TcpSocket-members.htm)

sf::TcpSocket Class Reference

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

Specialized socket using the TCP protocol. [More...](http://docs.google.com/classsf_1_1TcpSocket.htm#details)

#include <[TcpSocket.hpp](http://docs.google.com/TcpSocket_8hpp_source.htm)>

Inheritance diagram for sf::TcpSocket:



| Public Types | |
| --- | --- |
| enum | [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) {  [Done](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dca1de3a85bc56d3ae85b3d0f3cfd04ae90),  [NotReady](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dca8554848daae98f996e131bdeed076c09),  [Disconnected](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dcab215141f756acdc23c67fad149710eb1),  [Error](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dca1dc9854433a28c22e192721179a2df5d)  } |
|  | Status codes that may be returned by socket functions. [More...](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) |
|  | |
| enum | { [AnyPort](http://docs.google.com/classsf_1_1Socket.htm#a5deb2c955fd347259c3a20d27b2481aaa5a3c30fd128895403afc11076f461b19) = 0 } |
|  | Some special values used by sockets. [More...](http://docs.google.com/classsf_1_1Socket.htm#a5deb2c955fd347259c3a20d27b2481aa) |
|  | |

| Public Member Functions | |
| --- | --- |
|  | [TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm#a62a9bf81fd7f15fedb29fd1348483236) () |
|  | Default constructor. |
|  | |
| unsigned short | [getLocalPort](http://docs.google.com/classsf_1_1TcpSocket.htm#ab47eeb1cb71f2f251a83bc823773f1b3) () const |
|  | Get the port to which the socket is bound locally. |
|  | |
| [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) | [getRemoteAddress](http://docs.google.com/classsf_1_1TcpSocket.htm#a7904ca6ab9e018021e305a3aeb7a1b9a) () const |
|  | Get the address of the connected peer. |
|  | |
| unsigned short | [getRemotePort](http://docs.google.com/classsf_1_1TcpSocket.htm#abc05220e06f1522144cecab822e79296) () const |
|  | Get the port of the connected peer to which the socket is connected. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [connect](http://docs.google.com/classsf_1_1TcpSocket.htm#a68cd42d5ab70ab54b16787f555951c40) (const [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) &remoteAddress, unsigned short remotePort, [Time](http://docs.google.com/classsf_1_1Time.htm) timeout=[Time::Zero](http://docs.google.com/classsf_1_1Time.htm#a8db127b632fa8da21550e7282af11fa0)) |
|  | Connect the socket to a remote peer. |
|  | |
| void | [disconnect](http://docs.google.com/classsf_1_1TcpSocket.htm#ac18f518a9be3d6be5e74b9404c253c1e) () |
|  | Disconnect the socket from its remote peer. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [send](http://docs.google.com/classsf_1_1TcpSocket.htm#affce26ab3bcc4f5b9269dad79db544c0) (const void \*data, std::size\_t size) |
|  | Send raw data to the remote peer. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [receive](http://docs.google.com/classsf_1_1TcpSocket.htm#a90ce50811ea61d4f00efc62bb99ae1af) (void \*data, std::size\_t size, std::size\_t &received) |
|  | Receive raw data from the remote peer. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [send](http://docs.google.com/classsf_1_1TcpSocket.htm#a0f8276e2b1c75aac4a7b0a707b250f44) ([Packet](http://docs.google.com/classsf_1_1Packet.htm) &packet) |
|  | Send a formatted packet of data to the remote peer. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [receive](http://docs.google.com/classsf_1_1TcpSocket.htm#aa655352609bc9804f2baa020df3e7331) ([Packet](http://docs.google.com/classsf_1_1Packet.htm) &packet) |
|  | Receive a formatted packet of data from the remote peer. |
|  | |
| void | [setBlocking](http://docs.google.com/classsf_1_1Socket.htm#a165fc1423e281ea2714c70303d3a9782) (bool blocking) |
|  | Set the blocking state of the socket. |
|  | |
| bool | [isBlocking](http://docs.google.com/classsf_1_1Socket.htm#a0ec0d831b015e32eb5935fd2a9f8c67c) () const |
|  | Tell whether the socket is in blocking or non-blocking mode. |
|  | |

| Protected Types | |
| --- | --- |
| enum | [Type](http://docs.google.com/classsf_1_1Socket.htm#a5d3ff44e56e68f02816bb0fabc34adf8) {  [Tcp](http://docs.google.com/classsf_1_1Socket.htm#a5d3ff44e56e68f02816bb0fabc34adf8acc02e97e90234b957eaad4dff7f22214),  [Udp](http://docs.google.com/classsf_1_1Socket.htm#a5d3ff44e56e68f02816bb0fabc34adf8a6ebf3094830db4820191a327f3cc6ce2)  } |
|  | Types of protocols that the socket can use. [More...](http://docs.google.com/classsf_1_1Socket.htm#a5d3ff44e56e68f02816bb0fabc34adf8) |
|  | |

| Protected Member Functions | |
| --- | --- |
| SocketHandle | [getHandle](http://docs.google.com/classsf_1_1Socket.htm#ac0c63b13e61da8294bf54e888e97f9a3) () const |
|  | Return the internal handle of the socket. |
|  | |
| void | [create](http://docs.google.com/classsf_1_1Socket.htm#aafbe140f4b1921e0d19e88cf7a61dcbc) () |
|  | Create the internal representation of the socket. |
|  | |
| void | [create](http://docs.google.com/classsf_1_1Socket.htm#af1dd898f7aa3ead7ff7b2d1c20e97781) (SocketHandle handle) |
|  | Create the internal representation of the socket from a socket handle. |
|  | |
| void | [close](http://docs.google.com/classsf_1_1Socket.htm#a71f2f5c2aa99e01cafe824fee4c573be) () |
|  | Close the socket gracefully. |
|  | |

| Friends | |
| --- | --- |
| class | **TcpListener** |
|  | |

## Detailed Description

Specialized socket using the TCP protocol.

TCP is a connected protocol, which means that a TCP socket can only communicate with the host it is 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 informations 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 functions. 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](http://docs.google.com/classsf_1_1Packet.htm)), which are easier to use and provide more safety regarding the data that is exchanged. You can look at the [sf::Packet](http://docs.google.com/classsf_1_1Packet.htm) class to get more details about how they work.

The socket is automatically disconnected when it is destroyed, but if you want to explicitely close the connection while the socket instance is still alive, you can call disconnect.

Usage example:

// ----- The client -----

// Create a socket and connect it to 192.168.1.50 on port 55001

[sf::TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm) socket;

socket.[connect](http://docs.google.com/classsf_1_1TcpSocket.htm#a68cd42d5ab70ab54b16787f555951c40)("192.168.1.50", 55001);

// Send a message to the connected host

std::string message = "Hi, I am a client";

socket.[send](http://docs.google.com/classsf_1_1TcpSocket.htm#affce26ab3bcc4f5b9269dad79db544c0)(message.c\_str(), message.size() + 1);

// Receive an answer from the server

char buffer[1024];

std::size\_t received = 0;

socket.[receive](http://docs.google.com/classsf_1_1TcpSocket.htm#a90ce50811ea61d4f00efc62bb99ae1af)(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](http://docs.google.com/classsf_1_1TcpListener.htm) listener;

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

// Wait for a connection

[sf::TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm) socket;

listener.[accept](http://docs.google.com/classsf_1_1TcpListener.htm#ae2c83ce5a64d50b68180c46bef0a7346)(socket);

std::cout << "New client connected: " << socket.[getRemoteAddress](http://docs.google.com/classsf_1_1TcpSocket.htm#a7904ca6ab9e018021e305a3aeb7a1b9a)() << std::endl;

// Receive a message from the client

char buffer[1024];

std::size\_t received = 0;

socket.[receive](http://docs.google.com/classsf_1_1TcpSocket.htm#a90ce50811ea61d4f00efc62bb99ae1af)(buffer, sizeof(buffer), received);

std::cout << "The client said: " << buffer << std::endl;

// Send an answer

std::string message = "Welcome, client";

socket.[send](http://docs.google.com/classsf_1_1TcpSocket.htm#affce26ab3bcc4f5b9269dad79db544c0)(message.c\_str(), message.size() + 1);

See Also[sf::Socket](http://docs.google.com/classsf_1_1Socket.htm), [sf::UdpSocket](http://docs.google.com/classsf_1_1UdpSocket.htm), [sf::Packet](http://docs.google.com/classsf_1_1Packet.htm)

Definition at line [46](http://docs.google.com/TcpSocket_8hpp_source.htm#l00046) of file [TcpSocket.hpp](http://docs.google.com/TcpSocket_8hpp_source.htm).

## Member Enumeration Documentation

| | anonymous enum | | --- | | inherited |
| --- | --- | --- |

Some special values used by sockets.

**Enumerator:**

| *AnyPort* | Special value that tells the system to pick any available port. |
| --- | --- |

Definition at line [65](http://docs.google.com/Socket_8hpp_source.htm#l00065) of file [Socket.hpp](http://docs.google.com/Socket_8hpp_source.htm).

| | enum [sf::Socket::Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | | --- | | inherited |
| --- | --- | --- |

Status codes that may be returned by socket functions.

**Enumerator:**

| *Done* | The socket has sent / received the data. |
| --- | --- |
| *NotReady* | The socket is not ready to send / receive data yet. |
| *Disconnected* | The TCP socket has been disconnected. |
| *Error* | An unexpected error happened. |

Definition at line [53](http://docs.google.com/Socket_8hpp_source.htm#l00053) of file [Socket.hpp](http://docs.google.com/Socket_8hpp_source.htm).

| | enum [sf::Socket::Type](http://docs.google.com/classsf_1_1Socket.htm#a5d3ff44e56e68f02816bb0fabc34adf8) | | --- | | protectedinherited |
| --- | --- | --- |

Types of protocols that the socket can use.

**Enumerator:**

| *Tcp* | TCP protocol. |
| --- | --- |
| *Udp* | UDP protocol. |

Definition at line [113](http://docs.google.com/Socket_8hpp_source.htm#l00113) of file [Socket.hpp](http://docs.google.com/Socket_8hpp_source.htm).

## Constructor & Destructor Documentation

| sf::TcpSocket::TcpSocket | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Default constructor.

## Member Function Documentation

| | void sf::Socket::close | ( |  | ) |  | | --- | --- | --- | --- | --- | | protectedinherited |
| --- | --- | --- | --- | --- | --- | --- |

Close the socket gracefully.

This function can only be accessed by derived classes.

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::TcpSocket::connect | ( | const [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) & | *remoteAddress*, |
| --- | --- | --- | --- |
|  |  | unsigned short | *remotePort*, |
|  |  | [Time](http://docs.google.com/classsf_1_1Time.htm) | *timeout* = [Time::Zero](http://docs.google.com/classsf_1_1Time.htm#a8db127b632fa8da21550e7282af11fa0) |
|  | ) |  |  |

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 |

ReturnsStatus code See Also[disconnect](http://docs.google.com/classsf_1_1TcpSocket.htm#ac18f518a9be3d6be5e74b9404c253c1e)

| | void sf::Socket::create | ( |  | ) |  | | --- | --- | --- | --- | --- | | protectedinherited |
| --- | --- | --- | --- | --- | --- | --- |

Create the internal representation of the socket.

This function can only be accessed by derived classes.

| | void sf::Socket::create | ( | SocketHandle | *handle* | ) |  | | --- | --- | --- | --- | --- | --- | | protectedinherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

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, this function has no effect.

See Also[connect](http://docs.google.com/classsf_1_1TcpSocket.htm#a68cd42d5ab70ab54b16787f555951c40)

| | SocketHandle sf::Socket::getHandle | ( |  | ) | const | | --- | --- | --- | --- | --- | | protectedinherited |
| --- | --- | --- | --- | --- | --- | --- |

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.

ReturnsThe 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.

ReturnsPort to which the socket is bound See Also[connect](http://docs.google.com/classsf_1_1TcpSocket.htm#a68cd42d5ab70ab54b16787f555951c40), [getRemotePort](http://docs.google.com/classsf_1_1TcpSocket.htm#abc05220e06f1522144cecab822e79296)

| [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) sf::TcpSocket::getRemoteAddress | ( |  | ) | const |
| --- | --- | --- | --- | --- |

Get the address of the connected peer.

It the socket is not connected, this function returns [sf::IpAddress::None](http://docs.google.com/classsf_1_1IpAddress.htm#a4619b4abbe3c8fef056e7299db967404).

ReturnsAddress of the remote peer See Also[getRemotePort](http://docs.google.com/classsf_1_1TcpSocket.htm#abc05220e06f1522144cecab822e79296)

| 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.

ReturnsRemote port to which the socket is connected See Also[getRemoteAddress](http://docs.google.com/classsf_1_1TcpSocket.htm#a7904ca6ab9e018021e305a3aeb7a1b9a)

| | bool sf::Socket::isBlocking | ( |  | ) | const | | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- |

Tell whether the socket is in blocking or non-blocking mode.

ReturnsTrue if the socket is blocking, false otherwise See Also[setBlocking](http://docs.google.com/classsf_1_1Socket.htm#a165fc1423e281ea2714c70303d3a9782)

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) 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 |

ReturnsStatus code See Also[send](http://docs.google.com/classsf_1_1TcpSocket.htm#affce26ab3bcc4f5b9269dad79db544c0)

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::TcpSocket::receive | ( | [Packet](http://docs.google.com/classsf_1_1Packet.htm) & | *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](http://docs.google.com/classsf_1_1Packet.htm) to fill with the received data |
| --- | --- |

ReturnsStatus code See Also[send](http://docs.google.com/classsf_1_1TcpSocket.htm#affce26ab3bcc4f5b9269dad79db544c0)

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::TcpSocket::send | ( | const void \* | *data*, |
| --- | --- | --- | --- |
|  |  | std::size\_t | *size* |
|  | ) |  |  |

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 |

ReturnsStatus code See Also[receive](http://docs.google.com/classsf_1_1TcpSocket.htm#a90ce50811ea61d4f00efc62bb99ae1af)

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::TcpSocket::send | ( | [Packet](http://docs.google.com/classsf_1_1Packet.htm) & | *packet* | ) |  |
| --- | --- | --- | --- | --- | --- |

Send a formatted packet of data to the remote peer.

This function will fail if the socket is not connected.

Parameters

| packet | [Packet](http://docs.google.com/classsf_1_1Packet.htm) to send |
| --- | --- |

ReturnsStatus code See Also[receive](http://docs.google.com/classsf_1_1TcpSocket.htm#a90ce50811ea61d4f00efc62bb99ae1af)

| | void sf::Socket::setBlocking | ( | bool | *blocking* | ) |  | | --- | --- | --- | --- | --- | --- | | inherited |
| --- | --- | --- | --- | --- | --- | --- | --- |

Set the blocking state of the socket.

In blocking mode, calls will not return until they have completed their task. For example, a call to Receive in blocking mode won't return until some data was actually received. In non-blocking mode, calls will 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](http://docs.google.com/classsf_1_1Socket.htm#a0ec0d831b015e32eb5935fd2a9f8c67c)

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

* [TcpSocket.hpp](http://docs.google.com/TcpSocket_8hpp_source.htm)

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