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**
* [UdpSocket](http://docs.google.com/classsf_1_1UdpSocket.htm)

[Public Types](#_gjdgxs) | [Public Member Functions](#_30j0zll) | [Protected Types](#_1fob9te) | [Protected Member Functions](#_3znysh7) | [List of all members](http://docs.google.com/classsf_1_1UdpSocket-members.htm)

sf::UdpSocket Class Reference

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

Specialized socket using the UDP protocol. [More...](http://docs.google.com/classsf_1_1UdpSocket.htm#details)

#include <[UdpSocket.hpp](http://docs.google.com/UdpSocket_8hpp_source.htm)>

Inheritance diagram for sf::UdpSocket:



| Public Types | |
| --- | --- |
| enum | { [MaxDatagramSize](http://docs.google.com/classsf_1_1UdpSocket.htm#a14c7b7816e33ed1ef1f2fdb2404c06b5a728a7d33027bee0d65f70f964dd9c9eb) = 65507 } |
|  | |
| 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 | |
| --- | --- |
|  | [UdpSocket](http://docs.google.com/classsf_1_1UdpSocket.htm#abb10725e26dee9d3a8165fe87ffb71bb) () |
|  | Default constructor. |
|  | |
| unsigned short | [getLocalPort](http://docs.google.com/classsf_1_1UdpSocket.htm#a8ce625debd4b1f885366a69faa270086) () const |
|  | Get the port to which the socket is bound locally. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [bind](http://docs.google.com/classsf_1_1UdpSocket.htm#ab0bf8d32849836f92beb2fd734565481) (unsigned short port) |
|  | Bind the socket to a specific port. |
|  | |
| void | [unbind](http://docs.google.com/classsf_1_1UdpSocket.htm#a2c4abb8102a1bd31f51fcfe7f15427a3) () |
|  | Unbind the socket from the local port to which it is bound. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [send](http://docs.google.com/classsf_1_1UdpSocket.htm#a664ab8f26f37c21cc4de1b847c2efcca) (const void \*data, std::size\_t size, const [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) &remoteAddress, unsigned short remotePort) |
|  | Send raw data to a remote peer. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [receive](http://docs.google.com/classsf_1_1UdpSocket.htm#ade9ca0f7ed7919136917b0b997a9833a) (void \*data, std::size\_t size, std::size\_t &received, [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) &remoteAddress, unsigned short &remotePort) |
|  | Receive raw data from a remote peer. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [send](http://docs.google.com/classsf_1_1UdpSocket.htm#a48969a62c80d40fd74293a740798e435) ([Packet](http://docs.google.com/classsf_1_1Packet.htm) &packet, const [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) &remoteAddress, unsigned short remotePort) |
|  | Send a formatted packet of data to a remote peer. |
|  | |
| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) | [receive](http://docs.google.com/classsf_1_1UdpSocket.htm#afdd5c655d00c96222d5b477fc057a22b) ([Packet](http://docs.google.com/classsf_1_1Packet.htm) &packet, [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) &remoteAddress, unsigned short &remotePort) |
|  | Receive a formatted packet of data from a 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. |
|  | |

## Detailed Description

Specialized socket using the UDP protocol.

A UDP socket is a connectionless socket.

Instead of connecting once to a remote host, like TCP sockets, it can send to and receive from any host at any time.

It is a datagram protocol: bounded blocks of data (datagrams) are transfered over the network rather than a continuous stream of data (TCP). Therefore, one call to send will always match one call to receive (if the datagram is not lost), with the same data that was sent.

The UDP protocol is lightweight but unreliable. Unreliable means that datagrams may be duplicated, be lost or arrive reordered. However, if a datagram arrives, its data is guaranteed to be valid.

UDP is generally used for real-time communication (audio or video streaming, real-time games, etc.) where speed is crucial and lost data doesn't matter much.

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, whereas 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.

It is important to note that [UdpSocket](http://docs.google.com/classsf_1_1UdpSocket.htm) is unable to send datagrams bigger than MaxDatagramSize. In this case, it returns an error and doesn't send anything. This applies to both raw data and packets. Indeed, even packets are unable to split and recompose data, due to the unreliability of the protocol (dropped, mixed or duplicated datagrams may lead to a big mess when trying to recompose a packet).

If the socket is bound to a port, it is automatically unbound from it when the socket is destroyed. However, you can unbind the socket explicitely with the Unbind function if necessary, to stop receiving messages or make the port available for other sockets.

Usage example:

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

// Create a socket and bind it to the port 55001

[sf::UdpSocket](http://docs.google.com/classsf_1_1UdpSocket.htm) socket;

socket.[bind](http://docs.google.com/classsf_1_1UdpSocket.htm#ab0bf8d32849836f92beb2fd734565481)(55001);

// Send a message to 192.168.1.50 on port 55002

std::string message = "Hi, I am " + [sf::IpAddress::getLocalAddress](http://docs.google.com/classsf_1_1IpAddress.htm#a4c31622ad87edca48adbb8e8ed00ee4a)().[toString](http://docs.google.com/classsf_1_1IpAddress.htm#a52f4be92fb0ceb689abc469e4a85fd82)();

socket.[send](http://docs.google.com/classsf_1_1UdpSocket.htm#a664ab8f26f37c21cc4de1b847c2efcca)(message.c\_str(), message.size() + 1, "192.168.1.50", 55002);

// Receive an answer (most likely from 192.168.1.50, but could be anyone else)

char buffer[1024];

std::size\_t received = 0;

[sf::IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) sender;

unsigned short port;

socket.[receive](http://docs.google.com/classsf_1_1UdpSocket.htm#ade9ca0f7ed7919136917b0b997a9833a)(buffer, sizeof(buffer), received, sender, port);

std::cout << sender.ToString() << " said: " << buffer << std::endl;

// ----- The server -----

// Create a socket and bind it to the port 55002

[sf::UdpSocket](http://docs.google.com/classsf_1_1UdpSocket.htm) socket;

socket.[bind](http://docs.google.com/classsf_1_1UdpSocket.htm#ab0bf8d32849836f92beb2fd734565481)(55002);

// Receive a message from anyone

char buffer[1024];

std::size\_t received = 0;

[sf::IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) sender;

unsigned short port;

socket.[receive](http://docs.google.com/classsf_1_1UdpSocket.htm#ade9ca0f7ed7919136917b0b997a9833a)(buffer, sizeof(buffer), received, sender, port);

std::cout << sender.ToString() << " said: " << buffer << std::endl;

// Send an answer

std::string message = "Welcome " + sender.[toString](http://docs.google.com/classsf_1_1IpAddress.htm#a52f4be92fb0ceb689abc469e4a85fd82)();

socket.[send](http://docs.google.com/classsf_1_1UdpSocket.htm#a664ab8f26f37c21cc4de1b847c2efcca)(message.c\_str(), message.size() + 1, sender, port);

See Also[sf::Socket](http://docs.google.com/classsf_1_1Socket.htm), [sf::TcpSocket](http://docs.google.com/classsf_1_1TcpSocket.htm), [sf::Packet](http://docs.google.com/classsf_1_1Packet.htm)

Definition at line [45](http://docs.google.com/UdpSocket_8hpp_source.htm#l00045) of file [UdpSocket.hpp](http://docs.google.com/UdpSocket_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).

| anonymous enum |
| --- |

**Enumerator:**

| *MaxDatagramSize* | The maximum number of bytes that can be sent in a single UDP datagram. |
| --- | --- |

Definition at line [52](http://docs.google.com/UdpSocket_8hpp_source.htm#l00052) of file [UdpSocket.hpp](http://docs.google.com/UdpSocket_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::UdpSocket::UdpSocket | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Default constructor.

## Member Function Documentation

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::UdpSocket::bind | ( | unsigned short | *port* | ) |  |
| --- | --- | --- | --- | --- | --- |

Bind the socket to a specific port.

Binding the socket to a port is necessary for being able to receive data on that port. You can use the special value [Socket::AnyPort](http://docs.google.com/classsf_1_1Socket.htm#a5deb2c955fd347259c3a20d27b2481aaa5a3c30fd128895403afc11076f461b19) to tell the system to automatically pick an available port, and then call getLocalPort to retrieve the chosen port.

Parameters

| port | Port to bind the socket to |
| --- | --- |

ReturnsStatus code See Also[unbind](http://docs.google.com/classsf_1_1UdpSocket.htm#a2c4abb8102a1bd31f51fcfe7f15427a3), [getLocalPort](http://docs.google.com/classsf_1_1UdpSocket.htm#a8ce625debd4b1f885366a69faa270086)

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

Close the socket gracefully.

This function can only be accessed by derived classes.

| | 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 |
| --- | --- |

| | 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::UdpSocket::getLocalPort | ( |  | ) | const |
| --- | --- | --- | --- | --- |

Get the port to which the socket is bound locally.

If the socket is not bound to a port, this function returns 0.

ReturnsPort to which the socket is bound See Also[bind](http://docs.google.com/classsf_1_1UdpSocket.htm#ab0bf8d32849836f92beb2fd734565481)

| | 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::UdpSocket::receive | ( | void \* | *data*, |
| --- | --- | --- | --- |
|  |  | std::size\_t | *size*, |
|  |  | std::size\_t & | *received*, |
|  |  | [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) & | *remoteAddress*, |
|  |  | unsigned short & | *remotePort* |
|  | ) |  |  |

Receive raw data from a remote peer.

In blocking mode, this function will wait until some bytes are actually received. Be careful to use a buffer which is large enough for the data that you intend to receive, if it is too small then an error will be returned and *all* the data will be lost.

Parameters

| data | Pointer to the array to fill with the received bytes |
| --- | --- |
| size | Maximum number of bytes that can be received |
| received | This variable is filled with the actual number of bytes received |
| remoteAddress | Address of the peer that sent the data |
| remotePort | Port of the peer that sent the data |

ReturnsStatus code See Also[send](http://docs.google.com/classsf_1_1UdpSocket.htm#a664ab8f26f37c21cc4de1b847c2efcca)

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::UdpSocket::receive | ( | [Packet](http://docs.google.com/classsf_1_1Packet.htm) & | *packet*, |
| --- | --- | --- | --- |
|  |  | [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) & | *remoteAddress*, |
|  |  | unsigned short & | *remotePort* |
|  | ) |  |  |

Receive a formatted packet of data from a remote peer.

In blocking mode, this function will wait until the whole packet has been received.

Parameters

| packet | [Packet](http://docs.google.com/classsf_1_1Packet.htm) to fill with the received data |
| --- | --- |
| remoteAddress | Address of the peer that sent the data |
| remotePort | Port of the peer that sent the data |

ReturnsStatus code See Also[send](http://docs.google.com/classsf_1_1UdpSocket.htm#a664ab8f26f37c21cc4de1b847c2efcca)

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::UdpSocket::send | ( | const void \* | *data*, |
| --- | --- | --- | --- |
|  |  | std::size\_t | *size*, |
|  |  | const [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) & | *remoteAddress*, |
|  |  | unsigned short | *remotePort* |
|  | ) |  |  |

Send raw data to a remote peer.

Make sure that *size* is not greater than [UdpSocket::MaxDatagramSize](http://docs.google.com/classsf_1_1UdpSocket.htm#a14c7b7816e33ed1ef1f2fdb2404c06b5a728a7d33027bee0d65f70f964dd9c9eb), otherwise this function will fail and no data will be sent.

Parameters

| data | Pointer to the sequence of bytes to send |
| --- | --- |
| size | Number of bytes to send |
| remoteAddress | Address of the receiver |
| remotePort | Port of the receiver to send the data to |

ReturnsStatus code See Also[receive](http://docs.google.com/classsf_1_1UdpSocket.htm#ade9ca0f7ed7919136917b0b997a9833a)

| [Status](http://docs.google.com/classsf_1_1Socket.htm#a51bf0fd51057b98a10fbb866246176dc) sf::UdpSocket::send | ( | [Packet](http://docs.google.com/classsf_1_1Packet.htm) & | *packet*, |
| --- | --- | --- | --- |
|  |  | const [IpAddress](http://docs.google.com/classsf_1_1IpAddress.htm) & | *remoteAddress*, |
|  |  | unsigned short | *remotePort* |
|  | ) |  |  |

Send a formatted packet of data to a remote peer.

Make sure that the packet size is not greater than [UdpSocket::MaxDatagramSize](http://docs.google.com/classsf_1_1UdpSocket.htm#a14c7b7816e33ed1ef1f2fdb2404c06b5a728a7d33027bee0d65f70f964dd9c9eb), otherwise this function will fail and no data will be sent.

Parameters

| packet | [Packet](http://docs.google.com/classsf_1_1Packet.htm) to send |
| --- | --- |
| remoteAddress | Address of the receiver |
| remotePort | Port of the receiver to send the data to |

ReturnsStatus code See Also[receive](http://docs.google.com/classsf_1_1UdpSocket.htm#ade9ca0f7ed7919136917b0b997a9833a)

| | 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)

| void sf::UdpSocket::unbind | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Unbind the socket from the local port to which it is bound.

The port that the socket was previously using is immediately available after this function is called. If the socket is not bound to a port, this function has no effect.

See Also[bind](http://docs.google.com/classsf_1_1UdpSocket.htm#ab0bf8d32849836f92beb2fd734565481)

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

* [UdpSocket.hpp](http://docs.google.com/UdpSocket_8hpp_source.htm)

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