STUDY ON SOCKET PROGRAMMING

Date: 06-08-2024

# Given things to learn about:

1. Socket creation
2. TCP and UDP sockets programming
3. Study on system calls for socket programming
4. Methods used for sockets – listen, accept, connect, send , receive, sendTo, receiveFrom, close

## What are sockets?

A network socket is a software structure created within a network node of a inter process computer network that serves as an endpoint for sending and receiving data across the network. It’s structure and properties are defined by an API for the networking architecture. It’s lifetime lasts only till the execution of the process running in the node.

Socket provides a bidirectional First Come First Out communication. A socket connecting to the network is created at each end of the communication. Since we will have to both send and receive, each socket has it’s own address. This address is comprised of the IP address and a port number.

In a network, sockets are mostly used for a server-client communication. The server, when it is ready to send data, creates a socket and connects to the network port addresses. The client end, when it is ready to receive data, creates a socket and attempts to or sends a request to connect to the server socket.

1. **SOCKET CREATION:**

To create a socket in the specified family and of the specified type:

*s = socket(family, type, protocol);*

Breaking this down,

The family parameter describes the type of addresses the socket can communicate with. It is specified by one of the constants that are defined in *sys/socket.h*

Most common connection values are named as AF\_suite where AF means the address format to use in interpreting the names.

Some examples are: **AF\_APPLETALK**

Apple Computer Inc. Appletalk network

**AF\_INET6**

Internet family for IPv6 and IPv4

**AF\_INET**

Internet family for IPv4 only

**AF\_PUP**

Xerox Corporation PUP internet

**AF\_UNIX**

UNIX file system

Next, we have the type. A socket type describes the communication semantics and the transport protocol to be used. Socket types are defined in sys/socket.h. The types include  SOCK\_STREAM, SOCK\_DGRAM, or SOCK\_RAW, which are then supported by AF\_INET6, AF\_INET, and AF\_UNIX.

The types of sockets are

* **Stream Sockets** − Delivery is guaranteed. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries. In this type, only the first data will carry the receiver’s address, the other data will just follow the first one.

>> SOCK\_STREAM: This is for TCP (Transmission Control Protocol) sockets.

* **Datagram Sockets** − Delivery is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

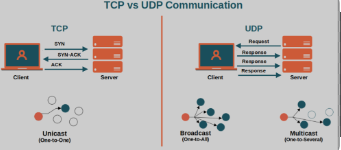
>> SOCK\_DGRAM: This is for UDP (User Datagram Protocol) sockets.

* **Raw Sockets** − These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

>> SOCK\_RAW: This is for raw sockets, which allow you to bypass the usual protocol layers and handle low-level protocols directly.

And lastly, we have protocol which is usually set to 0. The appropriate default one will be set based on the family and the type.

# TCP AND UDP SOCKETS AND THEIR CREATION:



TCP:

Transmission control protocol and user datagram protocol are protocols of the transport layer protocols.

TCP is a connection oriented protocol. It is one of the main protocols of the internet protocol suite. It lies between the application and network layers 🡪 which are used for providing reliable services. This helps in communication between different receives over the network. TCP tracks the segments being transmitted or received by assigning numbers to each and one of them. This enables the assurity of the data being reliable during transfer. TCP also has flow control which limits the rate at which the sender sends the data to ensure safe delivery. This also sends the data in a sequential order and is not dependent on the Operating system. It also allows many routing protocols and reduces the speed of data based on the speed of the receiver.

It is slower when compared to UDP. It is not suitable for LAN and PAN networks and does not have a multicast or a broadcast property.

Example of where it is used: **World Wide Web (WWW):, Email, File Transfer Protocol (FTP), Secure Shell (SSH), Streaming Media**

**UDP:**

**UDP is an unreliable network and is a connectionless protocol. It has no need to establish a connection as it can communicate with the server with the connect method. It helps in maintaining a low latency () and a loss tolerating connection in the network. It is used for simple request response communication especially when size of data is less and there is no need to worry about flow and error control. It supports packet switching and multicasting. It is used for some routing protocols such as RIP(routing information protocol). But it does not ensure the transferability of the data and does not send an acknowledgement with it.**

**Examples of where it used: Real-Time Multimedia Streaming, Online Gaming,** **DNS (Domain Name System) Queries,** **Network Monitoring,** **Multicasting,** **Routing Update Protocols**

TCP is **connection-oriented** which means it will **“setup” a connection** and then start transferring data. UDP is connectionless, which means it will just start sending and doesn’t care if it arrives or not. The connection that TCP will set up is called the **“3-way handshake,”**

**So in conclusion, they vary based on the usage and purpose. Now to create them,**

The **Select** function is used to select between TCP and UDP sockets. This function gives instructions to the kernel to wait for any of the multiple events to occur and awakens the process only after one or more events occur or a specified time passes.

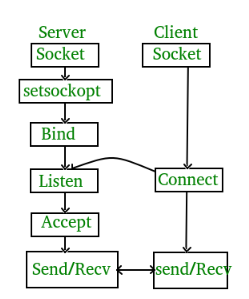
**TCP:**

**TCP Server –**

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

**TCP Client –**

1. Create TCP socket.
2. connect newly created client socket to server.



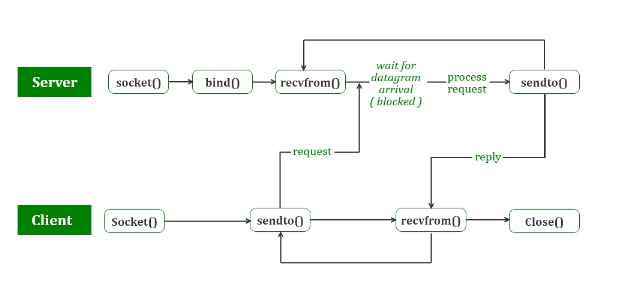
UDP:

**UDP Server :**

1. Create a UDP socket.
2. Bind the socket to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.
5. Go back to Step 3.

**UDP Client :**

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until a response from the server is received.
4. Process the reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.



# SOCKET PROGRAMMING:

The way of connecting the two nodes on a network to communicate and transfer data between them is called socket programming.

Socket()

Socket()

Bind() this is optional here

Bind()

Send and receive functions

Connect()

Send and receive functions

Close()

close

Accept()

Listen()

Since we have already covered on how to create a socket, let us now understand the next points:

### ****Setsockopt :****

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

*int setsockopt(int sockfd, int level, int optname,  const void \*optval, socklen\_t optlen);*

### ****Bind :****

**A socket when created has no name. A remote process will have to wait till an address is bound to the socket as it has no other way of referring to it.** In the Internet family, a connection is composed of local and remote addresses and local and remote ports. Duplicate ordered sets, such as:

*protocol, local address, local port, foreign address, foreign port*

cannot exist. In most families, connections must be unique.

The [bind(3SOCKET)](https://docs.oracle.com/cd/E23824_01/html/821-1466/index.html) interface enables a process to specifes the operating system to assign the local address of the socket. This interface forms the local address, local port set. [connect(3SOCKET)](https://docs.oracle.com/cd/E23824_01/html/821-1466/index.html) and [accept(3SOCKET)](https://docs.oracle.com/cd/E23824_01/html/821-1466/index.html) complete a socket's association by fixing the remote half of the address tuple. The [bind(3SOCKET)](https://docs.oracle.com/cd/E23824_01/html/821-1466/index.html) call is used as follows:

*bind (s, name, namelen);*

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

here sockfd is a file describer for the socket. Sockaddr is the address information that is necessary to bind the socket. Addrlen specifies the length of the address and lets the bind function know how many bytes to read from the address structure.

### ****Listen:****

The listen function in network programming is used to mark a socket as a passive socket that will be used to accept incoming connection requests. Here's a detailed explanation of the listen function and its parameters:

**sockfd**: This is the file descriptor for the socket that you want to listen on. It is the same socket descriptor that you obtained when you created the socket and bound it to an address using socket and bind, respectively.

**backlog**: This parameter specifies the maximum number of connections that can be queued up before the operating system starts rejecting new connections. It essentially sets a limit on how many connections the kernel can hold in the queue while the server is busy processing other requests.

### ****Accept:****

The accept function in network programming is used by a server to accept an incoming connection request on a listening socket.

*int new\_socket= accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);*

>>**sockfd**: This is the file descriptor of the listening socket. The socket must be in listening mode for accept to work.

>>**addr**: This is a pointer to a sockaddr structure where the address information of the connecting client will be stored. This structure is filled in by the accept function to provide details about the client’s address.

>>**addrlen**: This is a pointer to a socklen\_t variable that initially contains the size of the addr structure. Upon successful return from accept, this variable will contain the actual size of the client's address.

**Connect:**

The connect function is used in network programming to establish a connection to a remote server from a client socket

*int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);*

 **sockfd**: This is the file descriptor of the socket that you want to connect. It is obtained from a call to the socket function. The socket should be created with socket before calling connect.

 **addr**: This is a pointer to a sockaddr structure that contains the address information of the remote server you want to connect to. This structure is filled with the server's address and port information.

 **addrlen**: This specifies the length of the addr structure. It tells connect how many bytes to read from the address structure. It should be set to sizeof(struct sockaddr\_in) for IPv4 addresses or sizeof(struct sockaddr\_in6) for IPv6 addresses.

### Closing Sockets

A SOCK\_STREAM socket can be discarded by a [close(2)](https://docs.oracle.com/cd/E23824_01/html/821-1463/index.html) interface call. If data is queued to a socket that promises reliable delivery after a [close(2)](https://docs.oracle.com/cd/E23824_01/html/821-1463/index.html), the protocol continues to try to transfer the data. The data is discarded if it remains undelivered after an arbitrary period.

A [shutdown(3SOCKET)](https://docs.oracle.com/cd/E23824_01/html/821-1466/index.html) closes SOCK\_STREAM sockets gracefully. Both processes can acknowledge that they are no longer sending. This call has the form:

shutdown(s, how);

where how is defined as

**0**

Disallows further data reception

**1**

Disallows further data transmission

**2**

Disallows further transmission and further reception

The send() function is used to send data through a connected socket.

There are two versions of the API, as shown above. The base IBM® i API uses BSD 4.3 structures and syntax. The other uses syntax and structures compatible with the UNIX 98 programming interface specifications. You can select the UNIX 98 compatible interface with the [\_XOPEN\_SOURCE](https://www.ibm.com/docs/en/ssw_ibm_i_73/apis/_xopen_source.htm) macro.

## Parameters

**socket\_descriptor**

(Input) The socket descriptor that is to be written to.

**buffer**

(Input) The pointer to the buffer in which the data that is to be written is stored.

**buffer\_length**

(Input) The length of the buffer.

**flags**

(Input) A flag value that controls the transmission of the data. The flags value is either zero, or is obtained by performing an OR operation on the following constants:

|  |  |
| --- | --- |
| *MSG\_EOR* | Terminate a record, if supported by the protocol. |
| *MSG\_OOB* | Send data as out-of-band data. Valid only for sockets with an address family of AF\_INET or AF\_INET6 and type SOCK\_STREAM. |
| *MSG\_DONTROUTE* | Bypass routing. Valid only for sockets with address family of AF\_INET or AF\_INET6. It is ignored for other address families |

The sendto() function is used to send data through a connected or unconnected socket.

#include <sys/types.h>

#include <sys/socket.h>

int send(int socket\_descriptor,

char \*buffer,

int buffer\_length,

int flags)

There are two versions of the API, as shown above. The base IBM® i API uses BSD 4.3 structures and syntax. The other uses syntax and structures compatible with the UNIX 98 programming interface specifications. You can select the UNIX 98 compatible interface with the [\_XOPEN\_SOURCE](https://www.ibm.com/docs/en/ssw_ibm_i_73/apis/_xopen_source.htm) macro.

## Parameters

**socket\_descriptor**

(Input) The socket descriptor that is to be written to.

**buffer**

(Input) The pointer to the buffer in which the data that is to be written is stored.

**buffer\_length**

(Input) The length of the buffer.

**flags**

(Input) A flag value that controls the transmission of the data. The flags value is either zero, or is obtained by performing an OR operation on one or more of the following constants:

|  |  |
| --- | --- |
| *MSG\_EOR* | Terminate a record, if supported by the protocol. |
| *MSG\_OOB* | Send data as out-of-band data. Valid only for sockets with an address family of AF\_INET or AF\_INET6 and type SOCK\_STREAM. |
| *MSG\_DONTROUTE* | Bypass routing. Valid only for sockets with address family of AF\_INET or AF\_INET6. It is ignored for other address families. |

**destination\_address**

(Input) A pointer to a buffer of type **struct sockaddr** that contains the destination address to which the data is to be sent. The structure **sockaddr** is defined in **<sys/socket.h>**.

The BSD 4.3 structure is:

struct sockaddr {

u\_short sa\_family;

char sa\_data[14];

};

The BSD 4.4/UNIX 98 compatible structure is:

typedef uchar sa\_family\_t;

struct sockaddr {

uint8\_t sa\_len;

sa\_family\_t sa\_family;

char sa\_data[14];

};

The BSD 4.4 sa\_len field is the length of the address. The sa\_family field identifies the address family to which the address belongs, and sa\_data is the address whose format is dependent on the address family.

**address\_length**

(Input) The length of the destination\_address.

Syntax

#include <sys/types.h>

#include <sys/socket.h>

int sendto(int *socket\_descriptor*,

char \**buffer*,

int *buffer\_length*,

int *flags*,

struct sockaddr \**destination\_address*,

int *address\_length*)

# recv()--Receive Data

Syntax

#include <sys/types.h>

#include <sys/socket.h>

int recv(int *socket\_descriptor*,

char \**buffer*,

int *buffer\_length*,

int *flags*)

The recv() function is used to receive data through a socket.

There are two versions of the API, as shown above. The base IBM® i API uses BSD 4.3 structures and syntax. The other uses syntax and structures compatible with the UNIX 98 programming interface specifications. You can select the UNIX 98 compatible interface with the [\_XOPEN\_SOURCE](https://www.ibm.com/docs/en/ssw_ibm_i_73/apis/_xopen_source.htm) macro.

## Parameters

**socket\_descriptor**

(Input) The socket descriptor that is to be read from.

**buffer**

(Input) The pointer to the buffer in which the data that is to be read is stored.

**buffer\_length**

(Input) The length of the buffer.

**flags**

(Input) A flag value that controls the reception of the data. The flags value is either zero, or is obtained by performing an OR operation on one or more of the following constants:

|  |  |
| --- | --- |
| *MSG\_OOB* | Receive out-of-band data. Valid only for sockets with an address family of AF\_INET or AF\_INET6 and type SOCK\_STREAM. |
| *MSG\_PEEK* | Obtain a copy of the message without removing the message from the socket. |
| *MSG\_WAITALL* | Wait for a full request or an error. |

# recvfrom()--Receive Data

Syntax

#include <sys/types.h>

#include <sys/socket.h>

int recvfrom(int *socket\_descriptor*,

char \**buffer*,

int *buffer\_length*,

int *flags*,

struct sockaddr \**from\_address*,

int \**address\_length*)

The recvfrom() function is used to receive data through a connected or unconnected socket.

There are two versions of the API, as shown above. The base IBM® i API uses BSD 4.3 structures and syntax. The other uses syntax and structures compatible with the UNIX 98 programming interface specifications. You can select the UNIX 98 compatible interface with the [\_XOPEN\_SOURCE](https://www.ibm.com/docs/en/ssw_ibm_i_73/apis/_xopen_source.htm) macro.

## Parameters

**socket\_descriptor**

(Input) The socket descriptor that is to be read from.

**buffer**

(Input) The pointer to the buffer in which the data that is to be read is stored.

**buffer\_length**

(Input) The length of the buffer.

int **flags**

(Input) A flag value that controls the reception of the data. The flags value is either zero, or is obtained by performing an OR operation on one or more of the following constants:

|  |  |
| --- | --- |
| *MSG\_OOB* | Receive out-of-band data. Valid only for sockets with an address family of AF\_INET or AF\_INET6 and type SOCK\_STREAM. |
| *MSG\_PEEK* | Obtain a copy of the message without removing the message from the socket. |
| *MSG\_WAITALL* | Wait for a full request or an error. |

# Methods used for sockets:

1. Listen:

[Listen](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.listen?view=net-8.0) causes a connection-oriented [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) to listen for incoming connection attempts. The backlog parameter specifies the number of incoming connections that can be queued for acceptance. To determine the maximum number of connections you can specify, retrieve the [MaxConnections](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketoptionname?view=net-8.0" \l "system-net-sockets-socketoptionname-maxconnections) value. [Listen](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.listen?view=net-8.0) does not block.

*public void Listen (int backlog);’*

1. Accept:

Creates a new [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) for a newly created connection.

*public System.Net.Sockets.Socket Accept ();*

The `Accept` method retrieves the first pending connection request from a listening socket, creating and returning a new socket for that connection. This new socket is used solely for communication with the connected client, not for accepting further connections. In blocking mode, `Accept` waits for an incoming connection, while in non-blocking mode, it throws a `SocketException` if no requests are queued. Use the `SocketException.ErrorCode` property for specific error details and refer to the Windows Sockets version 2 API documentation for more information.

1. Connect

Establishes a connection to a remote host.

*public void Connect (System.Net.EndPoint remoteEP);*

*For TCP connections, the `Connect` method establishes a network connection between your local endpoint and the specified remote endpoint. Once connected, you can send and receive data using the `Send` and `Receive` methods.*

*With UDP, calling `Connect` is optional. Without it, you can use `SendTo` and `ReceiveFrom` to communicate with a remote host. If you do call `Connect`, it sets a default remote address, discarding any datagrams from other addresses. To use a broadcast address as the default, you must first set the `SocketOptionName.Broadcast` option using `SetSocketOption`, or `Connect` will throw a `SocketException`. For specific error codes, check `SocketException.ErrorCode` and refer to the Windows Sockets documentation.*

*By default, `Connect` is a blocking call. If you set the `Blocking` property to `false`, it may throw a `SocketException` for TCP connections due to the time required to establish the connection. Connectionless protocols won’t throw exceptions in this case. If you encounter a `SocketException`, use `SocketException.ErrorCode` to get the error code and check the Windows Sockets documentation for details. If you see `WSAEWOULDBLOCK`, the connection is still being established—use the `Poll` method to monitor its progress.*

1. Send:

Sends data to a connected [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) using the specified [SocketFlags](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketflags?view=net-8.0).

public int Send (ReadOnlySpan<byte> buffer, System.Net.Sockets.SocketFlags socketFlags, out System.Net.Sockets.SocketError errorCode);

[Send](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send?view=net-8.0) synchronously sends data to the remote host specified in the [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) or [Accept](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.accept?view=net-8.0) method and returns the number of bytes successfully sent. [Send](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send?view=net-8.0) can be used for both connection-oriented and connectionless protocols.

This overload requires a buffer that contains the data you want to send. The [SocketFlags](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketflags?view=net-8.0) value defaults to 0, the buffer offset defaults to 0, and the number of bytes to send defaults to the size of the buffer.

If you're using a connectionless protocol, you must call [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) before calling this method, or [Send](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send?view=net-8.0) will throw a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0). If you're using a connection-oriented protocol, you must either use [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) to establish a remote host connection, or use [Accept](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.accept?view=net-8.0) to accept an incoming connection.

1. Receive

Receives data from a bound [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) into a receive buffer, using the specified [SocketFlags](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketflags?view=net-8.0).

public int Receive (byte[] buffer, int offset, int size, System.Net.Sockets.SocketFlags socketFlags, out System.Net.Sockets.SocketError errorCode);

The [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) method reads data into the buffer parameter and returns the number of bytes successfully read. You can call [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) from both connection-oriented and connectionless sockets.

If you are using a connection-oriented protocol, you must either call [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) to establish a remote host connection, or [Accept](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.accept?view=net-8.0) to accept an incoming connection prior to calling [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0). The [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) method will only read data that arrives from the remote host established in the [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) or [Accept](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.accept?view=net-8.0) method. If you are using a connectionless protocol, you can also use the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method. [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) will allow you to receive data arriving from any host.

If no data is available for reading, the [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) method will block until data is available, unless a time-out value was set by using [Socket.ReceiveTimeout](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivetimeout?view=net-8.0). If the time-out value was exceeded, the [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) call will throw a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0). If you are in non-blocking mode, and there is no data available in the protocol stack buffer, the [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) method will complete immediately and throw a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0). An error occurred when attempting to access the socket. See Remarks below. You can use the [Available](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.available?view=net-8.0) property to determine if data is available for reading. When [Available](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.available?view=net-8.0) is non-zero, retry the receive operation.

If you are using a connection-oriented [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0), the [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) method will read as much data as is available, up to the number of bytes specified by the size parameter. If the remote host shuts down the [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) connection with the [Shutdown](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.shutdown?view=net-8.0) method, and all available data has been received, the [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) method will complete immediately and return zero bytes.

If you are using a connectionless [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0), [Receive](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=net-8.0) will read the first queued datagram from the destination address you specify in the [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) method. If the datagram you receive is larger than the size of the buffer parameter, buffer gets filled with the first part of the message, the excess data is lost and a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0) is thrown.

1. Sendto

Sends data to a specific endpoint.

public int SendTo (byte[] buffer, int offset, int size, System.Net.Sockets.SocketFlags socketFlags, System.Net.EndPoint remoteEP);

In this overload, if you specify the [DontRoute](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketflags?view=net-8.0" \l "system-net-sockets-socketflags-dontroute) flag as the socketflags parameter, the data you are sending will not be routed.

If you are using a connectionless protocol, you do not need to establish a default remote host with the [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) method prior to calling [SendTo](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.sendto?view=net-8.0). You only need to do this if you intend to call the [Send](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send?view=net-8.0) method. If you do call the [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) method prior to calling [SendTo](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.sendto?view=net-8.0), the remoteEP parameter will override the specified default remote host for that send operation only. You are also not required to call the [Bind](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.bind?view=net-8.0) method, because the underlying service provider will assign the most appropriate local network address and port number. If you need to identify the assigned local network address and port number, you can use the [LocalEndPoint](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.localendpoint?view=net-8.0) property after the [SendTo](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.sendto?view=net-8.0) method successfully completes.

Although intended for connectionless protocols, [SendTo](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.sendto?view=net-8.0) also works with connection-oriented protocols. If you are using a connection-oriented protocol, you must first establish a remote host connection by calling the [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) method or accept an incoming connection request using the [Accept](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.accept?view=net-8.0) method. If you do not establish or accept a remote host connection, [SendTo](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.sendto?view=net-8.0) will throw a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0). You can also establish a default remote host for a connectionless protocol prior to calling the [SendTo](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.sendto?view=net-8.0) method. In either of these cases, [SendTo](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.sendto?view=net-8.0) will ignore the remoteEP parameter and only send data to the connected or default remote host.

1. Recievefrom

Receives the specified number of bytes into the data buffer, using the specified [SocketFlags](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketflags?view=net-8.0), and stores the endpoint.

public int ReceiveFrom (byte[] buffer, int size, System.Net.Sockets.SocketFlags socketFlags, ref System.Net.EndPoint remoteEP);

The [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method reads data into the buffer parameter, returns the number of bytes successfully read, and captures the remote host endpoint from which the data was sent. This method is useful if you intend to receive connectionless datagrams from an unknown host or multiple hosts.

This overload only requires you to provide a receive buffer, the number of bytes you want to receive, the necessary [SocketFlags](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketflags?view=net-8.0), and an [EndPoint](https://learn.microsoft.com/en-us/dotnet/api/system.net.endpoint?view=net-8.0) that represents the remote host. The buffer offset defaults to 0.

With connectionless protocols, [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) will read the first enqueued datagram received into the local network buffer. If the datagram you receive is larger than the size of buffer, the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method will fill buffer with as much of the message as is possible, and throw a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0). If you are using an unreliable protocol, the excess data will be lost. If you are using a reliable protocol, the excess data will be retained by the service provider and you can retrieve it by calling the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method with a large enough buffer.

If no data is available for reading, the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method will block until data is available. If you are in non-blocking mode, and there is no data available in the protocol stack buffer, the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method will complete immediately and throw a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0). You can use the [Available](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.available?view=net-8.0) property to determine if data is available for reading. When [Available](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.available?view=net-8.0) is non-zero, retry the receive operation.

Although [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) is intended for connectionless protocols, you can use a connection-oriented protocol as well. If you choose to do so, you must first either establish a remote host connection by calling the [Connect](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connect?view=net-8.0) method or accept an incoming remote host connection by calling the [Accept](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.accept?view=net-8.0) method. If you do not establish or accept a connection before calling the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method, you will get a [SocketException](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketexception?view=net-8.0). You can also establish a default remote host for a connectionless protocol prior to calling the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method.

With connection-oriented sockets, [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) will read as much data as is available up to the number of bytes specified by the size parameter. If the remote host shuts down the [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) connection with the [Shutdown](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.shutdown?view=net-8.0) method, and all available data has been received, the [ReceiveFrom](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receivefrom?view=net-8.0) method will complete immediately and return zero bytes.

1. close

Closes the [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) connection and releases all associated resources.

public void Close ();

The [Close](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.close?view=net-8.0) method closes the remote host connection and releases all managed and unmanaged resources associated with the [Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0). Upon closing, the [Connected](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.connected?view=net-8.0) property is set to false.

For connection-oriented protocols, it is recommended that you call [Shutdown](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.shutdown?view=net-8.0) before calling the [Close](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.close?view=net-8.0) method. This ensures that all data is sent and received on the connected socket before it is closed.

If you need to call [Close](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.close?view=net-8.0) without first calling [Shutdown](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.shutdown?view=net-8.0), you can ensure that data queued for outgoing transmission will be sent by setting the [DontLinger](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketoptionname?view=net-8.0" \l "system-net-sockets-socketoptionname-dontlinger)[Socket](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0) option to false and specifying a non-zero time-out interval. [Close](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.close?view=net-8.0) will then block until this data is sent or until the specified time-out expires. If you set [DontLinger](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socketoptionname?view=net-8.0" \l "system-net-sockets-socketoptionname-dontlinger) to false and specify a zero time-out interval, [Close](https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.close?view=net-8.0) releases the connection and automatically discards outgoing queued data.