Socket Class

Reference

Definition

Namespace: System.Net.Sockets

Assembly: System.Net.Sockets.dll

Implements the Berkeley sockets interface.

C#

```
public class Socket : IDisposable
```

Inheritance Object → Socket

Implements IDisposable

Examples

Synchronous mode

The following example shows how the Socket class can be used to send data to an HTTP server, printing the ASCII response to the standard output. This example blocks the calling thread until the entire page is received.

C#

```
private static void SendHttpRequest(Uri? uri = null, int port = 80)
{
    uri ??= new Uri("http://example.com");

    // Construct a minimalistic HTTP/1.1 request
    byte[] requestBytes = Encoding.ASCII.GetBytes(@$"GET {uri.AbsoluteUri}
HTTP/1.0
Host: {uri.Host}
Connection: Close

");
```

```
    // Create and connect a dual-stack socket
    using Socket socket = new Socket(SocketType.Stream, ProtocolType.Tcp);
    socket.Connect(uri.Host, port);

    // Send the request.
    // For the tiny amount of data in this example, the first call to Send()
will likely deliver the buffer completely,
    // however this is not guaranteed to happen for larger real-life buffers.
    // The best practice is to iterate until all the data is sent.
    int bytesSent = 0;
    while (bytesSent < requestBytes.Length)
    {
        bytesSent += socket.Send(requestBytes, bytesSent, requestBytes.Length -
bytesSent, SocketFlags.None);
    }

    // Do minimalistic buffering assuming ASCII response
    byte[] responseBytes = new byte[256];
    char[] responseChars = new char[256];

    while (true)
    {
        int bytesReceived = socket.Receive(responseBytes);

        // Receiving 0 bytes means EOF has been reached
        if (bytesReceived == 0) break;

        // Convert byteCount bytes to ASCII characters using the 'respon-
seChars' buffer as destination
        int charCount = Encoding.ASCII.GetChars(responseBytes, 0, bytesRe-
ceived, responseChars, 0);

        // Print the contents of the 'responseChars' buffer to Console.Out
        Console.Out.Write(responseChars, 0, charCount);
    }
}
```

Asynchronous mode

The second example demonstrates the same HTTP GET scenario, using Task-based asynchronous API-s, while also forwarding a CancellationToken to the asynchronous methods, making the entire operation cancellable.

> **Tip**

Socket's async methods that do not take a **CancellationToken** typically return a **Task**, which is allocated on the heap. Cancellable overloads are always **ValueTask**-returning; using them helps reducing allocations in high-performance code.

C#

```
private static async Task SendHttpRequestAsync(Uri? uri = null, int port = 80,
CancellationToken cancellationToken = default)
{
    uri ??= new Uri("http://example.com");

    // Construct a minimalistic HTTP/1.1 request
    byte[] requestBytes = Encoding.ASCII.GetBytes(@$"GET {uri.AbsoluteUri}
HTTP/1.1
Host: {uri.Host}
Connection: Close

");

    // Create and connect a dual-stack socket
    using Socket socket = new Socket(SocketType.Stream, ProtocolType.Tcp);
    await socket.ConnectAsync(uri.Host, port, cancellationToken);

    // Send the request.
    // For the tiny amount of data in this example, the first call to
SendAsync() will likely deliver the buffer completely,
    // however this is not guaranteed to happen for larger real-life buffers.
    // The best practice is to iterate until all the data is sent.
    int bytesSent = 0;
    while (bytesSent < requestBytes.Length)
    {
        bytesSent += await socket.SendAsync(requestBytes.AsMemory(bytesSent),
SocketFlags.None);
    }

    // Do minimalistic buffering assuming ASCII response
    byte[] responseBytes = new byte[256];
    char[] responseChars = new char[256];

    while (true)
    {
        int bytesReceived = await socket.ReceiveAsync(responseBytes,
SocketFlags.None, cancellationToken);

        // Receiving 0 bytes means EOF has been reached
        if (bytesReceived == 0) break;

        // Convert byteCount bytes to ASCII characters using the 'respon-
seChars' buffer as destination
```

```
        int charCount = Encoding.ASCII.GetChars(responseBytes, 0, bytesRe-
ceived, responseChars, 0);

        // Print the contents of the 'responseChars' buffer to Console.Out
        await Console.Out.WriteAsync(responseChars.AsMemory(0, charCount), can-
cellationToken);
    }
}
```

Remarks

The Socket class provides a rich set of methods and properties for network communications. The Socket class allows you to perform both synchronous and asynchronous data transfer using any of the communication protocols listed in the ProtocolType enumeration.

The Socket class follows the .NET Framework naming pattern for asynchronous methods. For example, the synchronous Receive method corresponds to the asynchronous ReceiveAsync variants.

Use the following methods for synchronous operation mode.

- If you are using a connection-oriented protocol such as TCP, your server can listen for connections using the Listen method. The Accept method processes any incoming connection requests and returns a Socket that you can use to communicate data with the remote host. Use this returned Socket to call the Send or Receive method. Call the Bind method prior to calling the Listen method if you want to specify the local IP address and port number. Use a port number of zero if you want the underlying service provider to assign a free port for you. If you want to connect to a listening host, call the Connect method. To communicate data, call the Send or Receive method.

- If you are using a connectionless protocol such as UDP, you do not need to listen for connections at all. Call the ReceiveFrom method to accept any incoming datagrams. Use the SendTo method to send datagrams to a remote host.

To process communications asynchronously, use the following methods.

- If you are using a connection-oriented protocol such as TCP, use ConnectAsync to connect with a listening host. Use SendAsync or ReceiveAsync to communicate data asynchronously. Incoming connection requests can be processed using AcceptAsync.

- If you are using a connectionless protocol such as UDP, you can use SendToAsync to send datagrams, and ReceiveFromAsyncto receive datagrams.

If you perform multiple asynchronous operations on a socket, they do not necessarily complete in the order in which they are started.

When you are finished sending and receiving data, use the Shutdown method to disable the Socket. After calling Shutdown, call the Close method to release all resources associated with the Socket.

The Socket class allows you to configure your Socket using the SetSocketOption method. Retrieve these settings using the GetSocketOption method.

Constructors

Socket(AddressFamily, SocketType, ProtocolType)	Initializes a new instance of the Socket class using the specified address family, socket type and protocol.
Socket(SafeSocketHandle)	Initializes a new instance of the Socket class for the specified socket handle.
Socket(SocketInformation)	Initializes a new instance of the Socket class using the specified value returned from DuplicateAndClose(Int32).
Socket(SocketType, ProtocolType)	Initializes a new instance of the Socket class using the specified socket type and protocol. If the operating system supports IPv6, this constructor creates a dual-mode socket; otherwise, it creates an IPv4 socket.

Properties

AddressFamily	Gets the address family of the Socket.
Available	Gets the amount of data that has been received from the network and is available to be read.
Blocking	Gets or sets a value that indicates whether the Socket is in blocking mode.
Connected	Gets a value that indicates whether a Socket is connected to a remote host as of the last Send or Receive operation.

DontFragment	Gets or sets a value that specifies whether the Socket allows Internet Protocol (IP) datagrams to be fragmented.
DualMode	Gets or sets a value that specifies whether the Socket is a dual-mode socket used for both IPv4 and IPv6.
EnableBroadcast	Gets or sets a Boolean value that specifies whether the Socket can send broadcast packets.
ExclusiveAddressUse	Gets or sets a Boolean value that specifies whether the Socket allows only one process to bind to a port.
Handle	Gets the operating system handle for the Socket.
IsBound	Gets a value that indicates whether the Socket is bound to a specific local port.
LingerState	Gets or sets a value that specifies whether the Socket will delay closing a socket in an attempt to send all pending data.
LocalEndPoint	Gets the local endpoint.
MulticastLoopback	Gets or sets a value that specifies whether outgoing multicast packets are delivered to the sending application.
NoDelay	Gets or sets a Boolean value that specifies whether the stream Socket is using the Nagle algorithm.
OSSupportsIPv4	Indicates whether the underlying operating system and network adaptors support Internet Protocol version 4 (IPv4).
OSSupportsIPv6	Indicates whether the underlying operating system and network adaptors support Internet Protocol version 6 (IPv6).
OSSupportsUnixDomainSockets	Indicates whether the underlying operating system support the Unix domain sockets.
ProtocolType	Gets the protocol type of the Socket.
ReceiveBufferSize	Gets or sets a value that specifies the size of the receive buffer of the Socket.
ReceiveTimeout	Gets or sets a value that specifies the amount of time after which a synchronous Receive call will time out.
RemoteEndPoint	Gets the remote endpoint.
SafeHandle	Gets a SafeSocketHandle that represents the socket handle that the current Socket object encapsulates.

SendBufferSize	Gets or sets a value that specifies the size of the send buffer of the Socket.
SendTimeout	Gets or sets a value that specifies the amount of time after which a synchronous Send call will time out.
SocketType	Gets the type of the Socket.
SupportsIPv4	**Obsolete.** Gets a value indicating whether IPv4 support is available and enabled on the current host.
SupportsIPv6	**Obsolete.** Gets a value that indicates whether the Framework supports IPv6 for certain obsolete Dns members.
Ttl	Gets or sets a value that specifies the Time To Live (TTL) value of Internet Protocol (IP) packets sent by the Socket.
UseOnlyOverlappedIO	**Obsolete.** Gets or sets a value that specifies whether the socket should only use Overlapped I/O mode. On .NET 5+ (including .NET Core versions), the value is always `false`.

Methods

Accept()	Creates a new Socket for a newly created connection.
AcceptAsync()	Accepts an incoming connection.
AcceptAsync(CancellationToken)	Accepts an incoming connection.
AcceptAsync(Socket)	Accepts an incoming connection.
AcceptAsync(Socket, CancellationToken)	Accepts an incoming connection.
AcceptAsync(SocketAsyncEventArgs)	Begins an asynchronous operation to accept an incoming connection attempt.
BeginAccept(AsyncCallback, Object)	Begins an asynchronous operation to accept an incoming connection attempt.
BeginAccept(Int32, AsyncCallback, Object)	Begins an asynchronous operation to accept an incoming connection attempt and receives the first block of data sent by the

	client application.
BeginAccept(Socket, Int32, AsyncCallback, Object)	Begins an asynchronous operation to accept an incoming connection attempt from a specified socket and receives the first block of data sent by the client application.
BeginConnect(EndPoint, AsyncCallback, Object)	Begins an asynchronous request for a remote host connection.
BeginConnect(IPAddress, Int32, AsyncCallback, Object)	Begins an asynchronous request for a remote host connection. The host is specified by an IPAddress and a port number.
BeginConnect(IPAddress[], Int32, AsyncCallback, Object)	Begins an asynchronous request for a remote host connection. The host is specified by an IPAddress array and a port number.
BeginConnect(String, Int32, AsyncCallback, Object)	Begins an asynchronous request for a remote host connection. The host is specified by a host name and a port number.
BeginDisconnect(Boolean, AsyncCallback, Object)	Begins an asynchronous request to disconnect from a remote endpoint.
BeginReceive(Byte[], Int32, Int32, SocketFlags, AsyncCallback, Object)	Begins to asynchronously receive data from a connected Socket.
BeginReceive(Byte[], Int32, Int32, SocketFlags, SocketError, AsyncCallback, Object)	Begins to asynchronously receive data from a connected Socket.
BeginReceive(IList<ArraySegment<Byte>>, SocketFlags, AsyncCallback, Object)	Begins to asynchronously receive data from a connected Socket.
BeginReceive(IList<ArraySegment<Byte>>, SocketFlags, SocketError, AsyncCallback, Object)	Begins to asynchronously receive data from a connected Socket.
BeginReceiveFrom(Byte[], Int32, Int32, SocketFlags, EndPoint, AsyncCallback, Object)	Begins to asynchronously receive data from a specified network device.
BeginReceiveMessageFrom(Byte[], Int32, Int32, SocketFlags, EndPoint, AsyncCallback, Object)	Begins to asynchronously receive the specified number of bytes of data into the specified location of the data buffer, using the specified SocketFlags, and stores the endpoint and packet information.
BeginSend(Byte[], Int32, Int32, SocketFlags, AsyncCallback,	Sends data asynchronously to a connected Socket.

Object)	
BeginSend(Byte[], Int32, Int32, SocketFlags, SocketError, AsyncCallback, Object)	Sends data asynchronously to a connected Socket.
BeginSend(IList<ArraySegment<Byte>>, SocketFlags, AsyncCallback, Object)	Sends data asynchronously to a connected Socket.
BeginSend(IList<ArraySegment<Byte>>, SocketFlags, SocketError, AsyncCallback, Object)	Sends data asynchronously to a connected Socket.
BeginSendFile(String, AsyncCallback, Object)	Sends the file `fileName` to a connected Socket object using the UseDefaultWorkerThread flag.
BeginSendFile(String, Byte[], Byte[], TransmitFileOptions, AsyncCallback, Object)	Sends a file and buffers of data asynchronously to a connected Socket object.
BeginSendTo(Byte[], Int32, Int32, SocketFlags, EndPoint, AsyncCallback, Object)	Sends data asynchronously to a specific remote host.
Bind(EndPoint)	Associates a Socket with a local endpoint.
CancelConnectAsync(SocketAsyncEventArgs)	Cancels an asynchronous request for a remote host connection.
Close()	Closes the Socket connection and releases all associated resources.
Close(Int32)	Closes the Socket connection and releases all associated resources with a specified timeout to allow queued data to be sent.
Connect(EndPoint)	Establishes a connection to a remote host.
Connect(IPAddress, Int32)	Establishes a connection to a remote host. The host is specified by an IP address and a port number.
Connect(IPAddress[], Int32)	Establishes a connection to a remote host. The host is specified by an array of IP addresses and a port number.
Connect(String, Int32)	Establishes a connection to a remote host. The host is specified by a host name and a port number.
ConnectAsync(EndPoint)	Establishes a connection to a remote host.

ConnectAsync(EndPoint, CancellationToken)	Establishes a connection to a remote host.
ConnectAsync(IPAddress, Int32)	Establishes a connection to a remote host.
ConnectAsync(IPAddress, Int32, CancellationToken)	Establishes a connection to a remote host.
ConnectAsync(IPAddress[], Int32)	Establishes a connection to a remote host.
ConnectAsync(IPAddress[], Int32, CancellationToken)	Establishes a connection to a remote host.
ConnectAsync(SocketAsyncEventArgs)	Begins an asynchronous request for a connection to a remote host.
ConnectAsync(SocketType, ProtocolType, SocketAsyncEventArgs)	Begins an asynchronous request for a connection to a remote host.
ConnectAsync(String, Int32)	Establishes a connection to a remote host.
ConnectAsync(String, Int32, CancellationToken)	Establishes a connection to a remote host.
Disconnect(Boolean)	Closes the socket connection and allows reuse of the socket.
DisconnectAsync(Boolean, CancellationToken)	Disconnects a connected socket from the remote host.
DisconnectAsync(SocketAsyncEventArgs)	Begins an asynchronous request to disconnect from a remote endpoint.
Dispose()	Releases all resources used by the current instance of the Socket class.
Dispose(Boolean)	Releases the unmanaged resources used by the Socket, and optionally disposes of the managed resources.
DuplicateAndClose(Int32)	Duplicates the socket reference for the target process, and closes the socket for this process.
EndAccept(Byte[], IAsyncResult)	Asynchronously accepts an incoming connection attempt and creates a new Socket object to handle remote host communication. This method returns a buffer that contains the initial data transferred.

EndAccept(Byte[], Int32, IAsyncResult)	Asynchronously accepts an incoming connection attempt and creates a new Socket object to handle remote host communication. This method returns a buffer that contains the initial data and the number of bytes transferred.
EndAccept(IAsyncResult)	Asynchronously accepts an incoming connection attempt and creates a new Socket to handle remote host communication.
EndConnect(IAsyncResult)	Ends a pending asynchronous connection request.
EndDisconnect(IAsyncResult)	Ends a pending asynchronous disconnect request.
EndReceive(IAsyncResult)	Ends a pending asynchronous read.
EndReceive(IAsyncResult, SocketError)	Ends a pending asynchronous read.
EndReceiveFrom(IAsyncResult, EndPoint)	Ends a pending asynchronous read from a specific endpoint.
EndReceiveMessageFrom(IAsyncResult, SocketFlags, EndPoint, IPPacketInformation)	Ends a pending asynchronous read from a specific endpoint. This method also reveals more information about the packet than EndReceiveFrom(IAsyncResult, EndPoint).
EndSend(IAsyncResult)	Ends a pending asynchronous send.
EndSend(IAsyncResult, SocketError)	Ends a pending asynchronous send.
EndSendFile(IAsyncResult)	Ends a pending asynchronous send of a file.
EndSendTo(IAsyncResult)	Ends a pending asynchronous send to a specific location.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
Finalize()	Frees resources used by the Socket class.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetRawSocketOption(Int32, Int32, Span<Byte>)	Gets a socket option value using platform-specific level and name identifiers.
GetSocketOption(SocketOptionLevel, SocketOptionName)	Returns the value of a specified Socket option, represented as an object.

GetSocketOption(SocketOptionLevel, SocketOptionName, Byte[])	Returns the specified Socket option setting, represented as a byte array.
GetSocketOption(SocketOptionLevel, SocketOptionName, Int32)	Returns the value of the specified Socket option in an array.
GetType()	Gets the Type of the current instance. (Inherited from Object)
IOControl(Int32, Byte[], Byte[])	Sets low-level operating modes for the Socket using numerical control codes.
IOControl(IOControlCode, Byte[], Byte[])	Sets low-level operating modes for the Socket using the IOControlCode enumeration to specify control codes.
Listen()	Places a Socket in a listening state.
Listen(Int32)	Places a Socket in a listening state.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Poll(Int32, SelectMode)	Determines the status of the Socket.
Poll(TimeSpan, SelectMode)	Determines the status of the Socket.
Receive(Byte[])	Receives data from a bound Socket into a receive buffer.
Receive(Byte[], Int32, Int32, SocketFlags)	Receives the specified number of bytes from a bound Socket into the specified offset position of the receive buffer, using the specified SocketFlags.
Receive(Byte[], Int32, Int32, SocketFlags, SocketError)	Receives data from a bound Socket into a receive buffer, using the specified SocketFlags.
Receive(Byte[], Int32, SocketFlags)	Receives the specified number of bytes of data from a bound Socket into a receive buffer, using the specified SocketFlags.
Receive(Byte[], SocketFlags)	Receives data from a bound Socket into a receive buffer, using the specified SocketFlags.
Receive(IList<ArraySegment<Byte>>)	Receives data from a bound Socket into the list of receive buffers.
Receive(IList<ArraySegment<Byte>>, SocketFlags)	Receives data from a bound Socket into the list of receive buffers, using the specified SocketFlags.

Receive(IList<ArraySegment<Byte>>, SocketFlags, SocketError)	Receives data from a bound Socket into the list of receive buffers, using the specified SocketFlags.
Receive(Span<Byte>)	Receives data from a bound Socket into a receive buffer.
Receive(Span<Byte>, SocketFlags)	Receives data from a bound Socket into a receive buffer, using the specified SocketFlags.
Receive(Span<Byte>, SocketFlags, SocketError)	Receives data from a bound Socket into a receive buffer, using the specified SocketFlags.
ReceiveAsync(ArraySegment<Byte>)	Receives data from a connected socket.
ReceiveAsync(ArraySegment<Byte>, SocketFlags)	Receives data from a connected socket.
ReceiveAsync(IList<ArraySegment<Byte>>)	Receives data from a connected socket.
ReceiveAsync(IList<ArraySegment<Byte>>, SocketFlags)	Receives data from a connected socket.
ReceiveAsync(Memory<Byte>, CancellationToken)	Receives data from a connected socket.
ReceiveAsync(Memory<Byte>, SocketFlags, CancellationToken)	Receives data from a connected socket.
ReceiveAsync(SocketAsyncEventArgs)	Begins an asynchronous request to receive data from a connected Socket object.
ReceiveFrom(Byte[], EndPoint)	Receives a datagram into the data buffer and stores the endpoint.
ReceiveFrom(Byte[], Int32, Int32, SocketFlags, EndPoint)	Receives the specified number of bytes of data into the specified location of the data buffer, using the specified SocketFlags, and stores the endpoint.
ReceiveFrom(Byte[], Int32, SocketFlags, EndPoint)	Receives the specified number of bytes into the data buffer, using the specified SocketFlags, and stores the endpoint.
ReceiveFrom(Byte[], SocketFlags, EndPoint)	Receives a datagram into the data buffer, using the specified SocketFlags, and stores the endpoint.
ReceiveFrom(Span<Byte>, EndPoint)	Receives a datagram into the data buffer and stores the endpoint.

ReceiveFrom(Span<Byte>, SocketFlags, EndPoint)	Receives a datagram into the data buffer, using the specified SocketFlags, and stores the endpoint.
ReceiveFromAsync(ArraySegment<Byte>, EndPoint)	Receives data and returns the endpoint of the sending host.
ReceiveFromAsync(ArraySegment<Byte>, SocketFlags, EndPoint)	Receives data and returns the endpoint of the sending host.
ReceiveFromAsync(Memory<Byte>, EndPoint, CancellationToken)	Receives data and returns the endpoint of the sending host.
ReceiveFromAsync(Memory<Byte>, SocketFlags, EndPoint, CancellationToken)	Receives data and returns the endpoint of the sending host.
ReceiveFromAsync(SocketAsyncEventArgs)	Begins to asynchronously receive data from a specified network device.
ReceiveMessageFrom(Byte[], Int32, Int32, SocketFlags, EndPoint, IPPacketInformation)	Receives the specified number of bytes of data into the specified location of the data buffer, using the specified SocketFlags, and stores the endpoint and packet information.
ReceiveMessageFrom(Span<Byte>, SocketFlags, EndPoint, IPPacketInformation)	Receives the specified number of bytes of data into the specified location of the data buffer, using the specified `socketFlags`, and stores the endpoint and packet information.
ReceiveMessageFromAsync(ArraySegment<Byte>, EndPoint)	Receives data and returns additional information about the sender of the message.
ReceiveMessageFromAsync(ArraySegment<Byte>, SocketFlags, EndPoint)	Receives data and returns additional information about the sender of the message.
ReceiveMessageFromAsync(Memory<Byte>, EndPoint, CancellationToken)	Receives data and returns additional information about the sender of the message.
ReceiveMessageFromAsync(Memory<Byte>, SocketFlags, EndPoint, CancellationToken)	Receives data and returns additional information about the sender of the message.

ReceiveMessageFrom Async(SocketAsyncEventArgs)	Begins to asynchronously receive the specified number of bytes of data into the specified location in the data buffer, using the specified SocketFlags, and stores the endpoint and packet information.
Select(IList, IList, IList, Int32)	Determines the status of one or more sockets.
Select(IList, IList, IList, Time Span)	Determines the status of one or more sockets.
Send(Byte[])	Sends data to a connected Socket.
Send(Byte[], Int32, Int32, Socket Flags)	Sends the specified number of bytes of data to a connected Socket, starting at the specified offset, and using the specified SocketFlags.
Send(Byte[], Int32, Int32, Socket Flags, SocketError)	Sends the specified number of bytes of data to a connected Socket, starting at the specified offset, and using the specified SocketFlags.
Send(Byte[], Int32, SocketFlags)	Sends the specified number of bytes of data to a connected Socket, using the specified SocketFlags.
Send(Byte[], SocketFlags)	Sends data to a connected Socket using the specified SocketFlags.
Send(IList<Array Segment<Byte>>)	Sends the set of buffers in the list to a connected Socket.
Send(IList<Array Segment<Byte>>, SocketFlags)	Sends the set of buffers in the list to a connected Socket, using the specified SocketFlags.
Send(IList<Array Segment<Byte>>, SocketFlags, SocketError)	Sends the set of buffers in the list to a connected Socket, using the specified SocketFlags.
Send(ReadOnlySpan<Byte>)	Sends data to a connected Socket.
Send(ReadOnlySpan<Byte>, SocketFlags)	Sends data to a connected Socket using the specified SocketFlags.
Send(ReadOnlySpan<Byte>, SocketFlags, SocketError)	Sends data to a connected Socket using the specified SocketFlags.
SendAsync(Array Segment<Byte>)	Sends data on a connected socket.
SendAsync(Array Segment<Byte>, SocketFlags)	Sends data on a connected socket.

SendAsync(IList<ArraySegment<Byte>>)	Sends data on a connected socket.
SendAsync(IList<ArraySegment<Byte>>, SocketFlags)	Sends data on a connected socket.
SendAsync(ReadOnlyMemory<Byte>, CancellationToken)	Sends data on a connected socket.
SendAsync(ReadOnlyMemory<Byte>, SocketFlags, CancellationToken)	Sends data on a connected socket.
SendAsync(SocketAsyncEventArgs)	Sends data asynchronously to a connected Socket object.
SendFile(String)	Sends the file `fileName` to a connected Socket object with the UseDefaultWorkerThread transmit flag.
SendFile(String, Byte[], Byte[], TransmitFileOptions)	Sends the file `fileName` and buffers of data to a connected Socket object using the specified TransmitFileOptions value.
SendFile(String, ReadOnlySpan<Byte>, ReadOnlySpan<Byte>, TransmitFileOptions)	Sends the file `fileName` and buffers of data to a connected Socket object using the specified TransmitFileOptions value.
SendFileAsync(String, CancellationToken)	Sends the file `fileName` to a connected Socket object.
SendFileAsync(String, ReadOnlyMemory<Byte>, ReadOnlyMemory<Byte>, TransmitFileOptions, CancellationToken)	Sends the file `fileName` and buffers of data to a connected Socket object using the specified TransmitFileOptions value.
SendPacketsAsync(SocketAsyncEventArgs)	Sends a collection of files or in memory data buffers asynchronously to a connected Socket object.
SendTo(Byte[], EndPoint)	Sends data to the specified endpoint.
SendTo(Byte[], Int32, Int32, SocketFlags, EndPoint)	Sends the specified number of bytes of data to the specified endpoint, starting at the specified location in the buffer, and using the specified SocketFlags.
SendTo(Byte[], Int32, SocketFlags, EndPoint)	Sends the specified number of bytes of data to the specified endpoint using the specified SocketFlags.

SendTo(Byte[], SocketFlags, EndPoint)	Sends data to a specific endpoint using the specified SocketFlags.
SendTo(ReadOnlySpan<Byte>, EndPoint)	Sends data to the specified endpoint.
SendTo(ReadOnlySpan<Byte>, SocketFlags, EndPoint)	Sends data to a specific endpoint using the specified SocketFlags.
SendToAsync(ArraySegment<Byte>, EndPoint)	Sends data to the specified remote host.
SendToAsync(ArraySegment<Byte>, SocketFlags, EndPoint)	Sends data to the specified remote host.
SendToAsync(ReadOnlyMemory<Byte>, EndPoint, CancellationToken)	Sends data to the specified remote host.
SendToAsync(ReadOnlyMemory<Byte>, SocketFlags, EndPoint, CancellationToken)	Sends data to the specified remote host.
SendToAsync(SocketAsyncEventArgs)	Sends data asynchronously to a specific remote host.
SetIPProtectionLevel(IPProtectionLevel)	Sets the IP protection level on a socket.
SetRawSocketOption(Int32, Int32, ReadOnlySpan<Byte>)	Sets a socket option value using platform-specific level and name identifiers.
SetSocketOption(SocketOptionLevel, SocketOptionName, Boolean)	Sets the specified Socket option to the specified Boolean value.
SetSocketOption(SocketOptionLevel, SocketOptionName, Byte[])	Sets the specified Socket option to the specified value, represented as a byte array.
SetSocketOption(SocketOptionLevel, SocketOptionName, Int32)	Sets the specified Socket option to the specified integer value.
SetSocketOption(SocketOptionLevel, SocketOptionName, Object)	Sets the specified Socket option to the specified value, represented as an object.

Shutdown(SocketShutdown)	Disables sends and receives on a Socket.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Extension Methods

AcceptAsync(Socket)	Performs an asynchronous operation on to accept an incoming connection attempt on the socket.
AcceptAsync(Socket, Socket)	Performs an asynchronous operation on to accept an incoming connection attempt on the socket.
ConnectAsync(Socket, EndPoint)	Establishes a connection to a remote host.
ConnectAsync(Socket, EndPoint, CancellationToken)	Establishes a connection to a remote host.
ConnectAsync(Socket, IPAddress, Int32)	Establishes a connection to a remote host. The host is specified by an IP address and a port number.
ConnectAsync(Socket, IPAddress, Int32, CancellationToken)	Establishes a connection to a remote host, which is specified by an IP address and a port number.
ConnectAsync(Socket, IPAddress[], Int32)	Establishes a connection to a remote host. The host is specified by an array of IP addresses and a port number.
ConnectAsync(Socket, IPAddress[], Int32, CancellationToken)	Establishes a connection to a remote host, which is specified by an array of IP addresses and a port number.
ConnectAsync(Socket, String, Int32)	Establishes a connection to a remote host. The host is specified by a host name and a port number.
ConnectAsync(Socket, String, Int32, CancellationToken)	Establishes a connection to a remote host, which is specified by a host name and a port number.
ReceiveAsync(Socket, ArraySegment<Byte>, SocketFlags)	Receives data from a connected socket.
ReceiveAsync(Socket, IList<ArraySegment<Byte>>, SocketFlags)	Receives data from a connected socket.

ReceiveAsync(Socket, Memory<Byte>, SocketFlags, CancellationToken)	Receives data from a connected socket.
ReceiveFromAsync(Socket, ArraySegment<Byte>, SocketFlags, EndPoint)	Receives data from a specified network device.
ReceiveMessageFromAsync(Socket, ArraySegment<Byte>, SocketFlags, EndPoint)	Receives the specified number of bytes of data into the specified location of the data buffer, using the specified SocketFlags, and stores the endpoint and packet information.
SendAsync(Socket, ArraySegment<Byte>, SocketFlags)	Sends data to a connected socket.
SendAsync(Socket, IList<ArraySegment<Byte>>, SocketFlags)	Sends data to a connected socket.
SendAsync(Socket, ReadOnlyMemory<Byte>, SocketFlags, CancellationToken)	Sends data to a connected socket.
SendToAsync(Socket, ArraySegment<Byte>, SocketFlags, EndPoint)	Sends data asynchronously to a specific remote host.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0
Xamarin.iOS	10.8
Xamarin.Mac	3.0

Thread Safety

It is safe to perform a send and a receive operation simultaneously on a Socket instance, but it's not recommended to issue multiple send or multiple receive calls concurrently. Depending on the underlying platform implementation, this may lead to unintended data interleaving for large or multi-buffer sends or receives.

See also

- System.Net
- System.Net.Cache
- System.Net.Security
- SocketAsyncEventArgs
- Network Programming in the .NET Framework
- Best Practices for System.Net Classes
- Cache Management for Network Applications
- Internet Protocol Version 6
- Network Programming Samples
- Network Tracing in the .NET Framework
- Security in Network Programming
- Socket Performance Enhancements in Version 3.5