System.Net.Sockets.Socket Class

```
[ILASM]
.class public Socket extends System.Object implements
System.IDisposable

[C#]
public class Socket: IDisposable
```

Assembly Info:

- *Name:* System
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 1.0.x.x
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- **System.IDisposable**

Summary

Creates a communication endpoint through which an application sends or receives data across a network.

Inherits From: System.Object

Library: Networking

Thread Safety: All public static members of this type are safe for multithreaded operations. No instance members are guaranteed to be thread safe.

Description

This class enables a **System.Net.Sockets.Socket** instance to communicate with another socket across a network. The communication can be through connection-oriented and connectionless protocols using either data streams or datagrams (discrete message packets).

Message-oriented protocols preserve message boundaries and require that for each **System.Net.Sockets.Socket.Send** method call there is one corresponding **System.Net.Sockets.Socket.Receive** method call. For stream-oriented protocols, data is transmitted without regards to message boundaries. In this case, for example, multiple **System.Net.Sockets.Socket.Receive** method calls may be necessary to retrieve all the data from one

System.Net.Sockets.Socket.Send method call. The protocol is set in the **Socket** class constructor.

A **System.Net.Sockets.Socket** instance has a local and a remote endpoint associated with it. The local endpoint contains the connection information for the current socket instance. The remote endpoint contains the connection information for the socket that the current instance communicates with. The endpoints are required to be an instance of a type derived from the **System.Net.EndPoint** class. For the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) protocols, an endpoint includes the address family, an Internet Protocol (IP) address, and a port number. For connection-oriented protocols (for example, TCP), the remote endpoint does not have to be specified when transferring data. For connectionless protocols (for example, UDP), the remote endpoint is required to be specified.

Methods are provided for both synchronous and asynchronous operations. A synchronous method can operate in blocking mode, in which it waits (blocks) until the operation is complete before returning, or in non-blocking mode, where it returns immediately, possibly before the operation has completed. The blocking mode is set through the **System.Net.Sockets.Socket.Blocking** property.

An asynchronous method returns immediately and, by convention, relies on a delegate to complete the operation. Asynchronous methods have names which correspond to their synchronous counterparts prefixed with either 'Begin' or End'. For example, the synchronous **System.Net.Sockets.Socket.Accept** method has asynchronous counterpart methods named **System.Net.Sockets.Socket.BeginAccept** and **System.Net.Sockets.Socket.EndAccept**. The example for the **System.Net.Sockets.Socket.BeginAccept** method shows the basic steps for using an asynchronous operation. A complete working example follows this discussion.

Connection-oriented protocols commonly use the client/server model. In this model, one of the sockets is set up as a server, and one or more sockets are set up as clients. A general procedure demonstrating the synchronous communication process for this model is as follows.

On the server-side:

1. Create a socket to listen for incoming connection requests.
2. Set the local endpoint using the **System.Net.Sockets.Socket.Bind** method.
3. Put the socket in the listening state using the **System.Net.Sockets.Socket.Listen** method.
4. At this point incoming connection requests from a client are placed in a queue.
5. Use the **System.Net.Sockets.Socket.Accept** method to create a server socket for a connection request issued by a client-side socket. This sets the remote endpoint.

6. Use the **System.Net.Sockets.Socket.Send** and **System.Net.Sockets.Socket.Receive** methods to communicate with the client socket.
7. When communication is finished, terminate the connection using the **System.Net.Sockets.Socket.Shutdown** method.
8. Release the resources allocated by the server socket using the **System.Net.Sockets.Socket.Close** method.
9. Release the resources allocated by the listener socket using the **System.Net.Sockets.Socket.Close** method.

On the client-side:

1. Create the client socket.
2. Connect to the server socket using the **System.Net.Sockets.Socket.Connect** method. This sets both the local and remote endpoints for the client socket.
3. Use the **System.Net.Sockets.Socket.Send** and **System.Net.Sockets.Socket.Receive** methods to communicate with the server socket.
4. > When communication is finished, terminate the connection using the **System.Net.Sockets.Socket.Shutdown** method.
5. Release the resources allocated by the client socket using the **System.Net.Sockets.Socket.Close** method.

The shutdown step in the previous procedure is not necessary but ensures that any pending data is not lost. If the **System.Net.Sockets.Socket.Shutdown** method is not called, the **System.Net.Sockets.Socket.Close** method shuts down the connection either gracefully or by force. A graceful closure attempts to transfer all pending data before the connection is terminated. Use the **System.Net.Sockets.SocketOptionName.Linger** socket option to specify a graceful closure for a socket.

[*Note:* This implementation is based on the UNIX sockets implementation in the Berkeley Software Distribution (BSD, release 4.3) from the University of California at Berkeley.]

Example

The following examples provide a client/server application that demonstrates the use of asynchronous communication between sockets. Run the client and server on different consoles.

The following code is for the server application. Start this application before the client application.

[C#]

```
using System;
using System.Threading;
using System.Text;
using System.Net;
using System.Net.Sockets;

public class Server
{
  // used to pass state information to delegate
  internal class StateObject
  {
    internal byte[] sBuffer;
    internal Socket sSocket;
    internal StateObject(int size, Socket sock) {
      sBuffer = new byte[size];
      sSocket = sock;
    }
  }
  static void Main()
  {
    IPAddress ipAddress =
      Dns.Resolve(Dns.GetHostName()).AddressList[0];

    IPEndPoint ipEndpoint =
      new IPEndPoint(ipAddress, 1800);

    Socket listenSocket =
      new Socket(AddressFamily.InterNetwork,
                 SocketType.Stream,
                 ProtocolType.Tcp);

    listenSocket.Bind(ipEndpoint);
    listenSocket.Listen(1);
    IAsyncResult asyncAccept = listenSocket.BeginAccept(
      new AsyncCallback(Server.acceptCallback),
      listenSocket);

    // could call listenSocket.EndAccept(asyncAccept) here
    // instead of in the callback method, but since
    // EndAccept blocks, the behavior would be similar to
    // calling the synchronous Accept method

    Console.Write("Connection in progress.");
    if(writeDot(asyncAccept) == true)
    {
      // allow time for callbacks to
      // finish before the program ends
      Thread.Sleep(3000);
    }
  }

  public static void
    acceptCallback(IAsyncResult asyncAccept) {
      Socket listenSocket = (Socket)asyncAccept.AsyncState;
      Socket serverSocket =
        listenSocket.EndAccept(asyncAccept);

      // arriving here means the operation completed
      // (asyncAccept.IsCompleted = true) but not
      // necessarily successfully
```

```
        if(serverSocket.Connected == false)
        {
          Console.WriteLine(".server is not connected.");
          return;
        }
        else Console.WriteLine(".server is connected.");

        listenSocket.Close();

        StateObject stateObject =
          new StateObject(16, serverSocket);

        // this call passes the StateObject because it
        // needs to pass the buffer as well as the socket
        IAsyncResult asyncReceive =
          serverSocket.BeginReceive(
            stateObject.sBuffer,
            0,
            stateObject.sBuffer.Length,
            SocketFlags.None,
            new AsyncCallback(receiveCallback),
            stateObject);

        Console.Write("Receiving data.");
        writeDot(asyncReceive);
    }

    public static void
      receiveCallback(IAsyncResult asyncReceive) {
        StateObject stateObject =
          (StateObject)asyncReceive.AsyncState;
        int bytesReceived =
          stateObject.sSocket.EndReceive(asyncReceive);

        Console.WriteLine(
          ".{0} bytes received: {1}",
          bytesReceived.ToString(),
          Encoding.ASCII.GetString(stateObject.sBuffer));

        byte[] sendBuffer =
          Encoding.ASCII.GetBytes("Goodbye");
        IAsyncResult asyncSend =
          stateObject.sSocket.BeginSend(
            sendBuffer,
            0,
            sendBuffer.Length,
            SocketFlags.None,
            new AsyncCallback(sendCallback),
            stateObject.sSocket);

        Console.Write("Sending response.");
        writeDot(asyncSend);
    }

    public static void sendCallback(IAsyncResult asyncSend) {
      Socket serverSocket = (Socket)asyncSend.AsyncState;
      int bytesSent = serverSocket.EndSend(asyncSend);
      Console.WriteLine(
        ".{0} bytes sent.{1}{1}Shutting down.",
        bytesSent.ToString(),
        Environment.NewLine);

      serverSocket.Shutdown(SocketShutdown.Both);
      serverSocket.Close();
    }

    // times out after 20 seconds but operation continues
    internal static bool writeDot(IAsyncResult ar)
    {
      int i = 0;
      while(ar.IsCompleted == false)
      {
        if(i++ > 40)
        {
          Console.WriteLine("Timed out.");
          return false;
        }
        Console.Write(".");
        Thread.Sleep(500);
      }
      return true;
    }
  }
```

The following code is for the client application. When starting the application, supply the hostname of the console running the server application as an input parameter (for example, ProgramName *hostname*).

[C#]

```
using System;
using System.Threading;
using System.Text;
using System.Net;
using System.Net.Sockets;

public class Client {

  // used to pass state information to delegate
  class StateObject
  {
    internal byte[] sBuffer;
    internal Socket sSocket;
    internal StateObject(int size, Socket sock) {
      sBuffer = new byte[size];
      sSocket = sock;
    }
  }

  static void Main(string[] argHostName)
  {
    IPAddress ipAddress =
      Dns.Resolve(argHostName[0]).AddressList[0];

    IPEndPoint ipEndpoint =
      new IPEndPoint(ipAddress, 1800);

    Socket clientSocket = new Socket(
      AddressFamily.InterNetwork,
      SocketType.Stream,
      ProtocolType.Tcp);

    IAsyncResult asyncConnect = clientSocket.BeginConnect(
      ipEndpoint,
      new AsyncCallback(connectCallback),
      clientSocket);

    Console.Write("Connection in progress.");
    if(writeDot(asyncConnect) == true)
    {
      // allow time for callbacks to
      // finish before the program ends
      Thread.Sleep(3000);
    }
  }

  public static void
    connectCallback(IAsyncResult asyncConnect) {
      Socket clientSocket =
        (Socket)asyncConnect.AsyncState;
      clientSocket.EndConnect(asyncConnect);
      // arriving here means the operation completed
      // (asyncConnect.IsCompleted = true) but not
      // necessarily successfully
      if(clientSocket.Connected == false)
      {
        Console.WriteLine(".client is not connected.");
        return;
      }
      else Console.WriteLine(".client is connected.");

      byte[] sendBuffer = Encoding.ASCII.GetBytes("Hello");
      IAsyncResult asyncSend = clientSocket.BeginSend(
        sendBuffer,
        0,
        sendBuffer.Length,
        SocketFlags.None,
        new AsyncCallback(sendCallback),
        clientSocket);

      Console.Write("Sending data.");
      writeDot(asyncSend);
```

```
        }

        public static void sendCallback(IAsyncResult asyncSend)
        {
          Socket clientSocket = (Socket)asyncSend.AsyncState;
          int bytesSent = clientSocket.EndSend(asyncSend);
          Console.WriteLine(
            ".{0} bytes sent.",
            bytesSent.ToString());

          StateObject stateObject =
            new StateObject(16, clientSocket);

          // this call passes the StateObject because it
          // needs to pass the buffer as well as the socket
          IAsyncResult asyncReceive =
            clientSocket.BeginReceive(
              stateObject.sBuffer,
              0,
              stateObject.sBuffer.Length,
              SocketFlags.None,
              new AsyncCallback(receiveCallback),
              stateObject);

          Console.Write("Receiving response.");
          writeDot(asyncReceive);
        }

        public static void
          receiveCallback(IAsyncResult asyncReceive) {
            StateObject stateObject =
             (StateObject)asyncReceive.AsyncState;

            int bytesReceived =
              stateObject.sSocket.EndReceive(asyncReceive);

            Console.WriteLine(
              ".{0} bytes received: {1}{2}{2}Shutting down.",
              bytesReceived.ToString(),
              Encoding.ASCII.GetString(stateObject.sBuffer),
              Environment.NewLine);

            stateObject.sSocket.Shutdown(SocketShutdown.Both);
            stateObject.sSocket.Close();
        }

        // times out after 2 seconds but operation continues
        internal static bool writeDot(IAsyncResult ar)
        {
          int i = 0;
          while(ar.IsCompleted == false)
          {
            if(i++ > 20)
            {
              Console.WriteLine("Timed out.");
              return false;
            }
```

```
      Console.Write(".");
      Thread.Sleep(100);
    }
    return true;
  }
}
```

The output of the server application is

```
Connection in progress...........server is connected.


Receiving data......5 bytes received: Hello


Sending response....7 bytes sent.


Shutting down.
```

```
-----------------------------------------
```

The output of the client application is

```
Connection in progress......client is connected.
```

```
Sending data......5 bytes sent.

Receiving response......7 bytes received: Goodbye

Shutting down.
```

Socket(System.Net.Sockets.AddressFamily, System.Net.Sockets.SocketType, System.Net.Sockets.ProtocolType) Constructor

```
[ILASM]
public rtspecialname specialname instance void
.ctor(valuetype System.Net.Sockets.AddressFamily
addressFamily, valuetype System.Net.Sockets.SocketType
socketType, valuetype System.Net.Sockets.ProtocolType
protocolType)

[C#]
public Socket(AddressFamily addressFamily, SocketType
socketType, ProtocolType protocolType)
```

Summary

Constructs and initializes a new instance of the **System.Net.Sockets.Socket** class.

Parameters

Parameter	Description
addressFamily	One of the values defined in the **System.Net.Sockets.AddressFamily** enumeration.
socketType	One of the values defined in the **System.Net.Sockets.SocketType** enumeration.
protocolType	One of the values defined in the **System.Net.Sockets.ProtocolType** enumeration.

Description

The *addressFamily* parameter specifies the addressing scheme used by the current instance, the *socketType* parameter specifies the socket type of the current instance, and the *protocolType* parameter specifies the protocol used by the current instance. The three parameters are not independent. Some address families restrict which protocols are used, and often the socket type is determined by the protocol. When the specified values are not a valid combination, a **System.Net.Sockets.SocketException** exception is thrown.

Using the **Unknown** member of either the **System.Net.Sockets.AddressFamily** or **System.Net.Sockets.ProtocolType** enumeration, results in a **System.Net.Sockets.SocketException** exception being thrown.

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	The combination of *addressFamily*, *socketType*, and *protocolType* is invalid. -or- An error occurred while creating the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]

Socket.Accept() Method

```
[ILASM]
.method public hidebysig instance class
System.Net.Sockets.Socket Accept()

[C#]
public Socket Accept()
```

Summary

Creates and initializes a new **System.Net.Sockets.Socket** instance and connects it to an incoming connection request.

Return Value

A new connected **System.Net.Sockets.Socket** instance.

Description

This method is used only on the server-side of connection-oriented protocols. It extracts the first connection request from the queue of pending requests, creates a new **System.Net.Sockets.Socket** instance, and connects this instance to the socket associated with the request.

The **System.Net.Sockets.Socket.Blocking** property of the socket determines the behavior of this method when there are no pending connection requests. When **false**, this method returns **null**. When **true**, this method blocks.

The following properties of the new **System.Net.Sockets.Socket** instance returned by this method have values identical to the corresponding properties of the current instance:

- **System.Net.Sockets.Socket.AddressFamily**
- **System.Net.Sockets.Socket.Blocking**
- **System.Net.Sockets.Socket.LocalEndPoint**
- **System.Net.Sockets.Socket.ProtocolType**
- **System.Net.Sockets.Socket.SocketType**

The **System.Net.Sockets.Socket.RemoteEndPoint** property of the new instance is set to the local endpoint of the first request in the input queue. The **System.Net.Sockets.Socket.Connected** property is set to **true**.

Exceptions

Exception	Condition
System.ArgumentException	An error occurred while creating the new **System.Net.Sockets.Socket**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.BeginAccept(System.AsyncCallback, System.Object) Method

[ILASM]

```
.method public hidebysig instance class System.IAsyncResult
BeginAccept(class System.AsyncCallback callback, object
state)
```

[C#]

```
public IAsyncResult BeginAccept(AsyncCallback callback,
object state)
```

Summary

Begins an asynchronous operation to accept an incoming connection request.

Parameters

Parameter	Description
callback	A **System.AsyncCallback** delegate, or **null**.
state	An application-defined object, or **null**.

Return Value

A **System.IAsyncResult** instance that contains information about the asynchronous operation.

Description

To retrieve the results of the operation and release resources allocated by the **System.Net.Sockets.Socket.BeginAccept** method, call the **System.Net.Sockets.Socket.EndAccept** method, and specify the **System.IAsyncResult** object returned by this method.

[*Note:* The **System.Net.Sockets.Socket.EndAccept** method should be called exactly once for each call to the **System.Net.Sockets.Socket.BeginAccept** method.]

If the *callback* parameter is not **null**, the method referenced by *callback* is invoked when the asynchronous operation completes. The **System.IAsyncResult** object returned by this method is passed as the argument to the method referenced by *callback*. The method referenced by *callback* can retrieve the results of the operation by calling the **System.Net.Sockets.Socket.EndAccept** method.

The *state* parameter can be any object that the caller wishes to have available for the duration of the asynchronous operation. This object is available via the **System.IAsyncResult.AsyncState** property of the object returned by this method.

To determine the connection status, check the **System.Net.Sockets.Socket.Connected** property, or use either the **System.Net.Sockets.Socket.Poll** or **System.Net.Sockets.Socket.Select** method.

[*Note:* For more information, see **System.Net.Sockets.Socket.Accept**, the synchronous version of this method.]

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while starting the operation. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

The following excerpt from the **System.Net.Sockets.Socket** class overview example outlines an asynchronous accept operation.

[C#]

```
public class Server
{
  static void Main()
  {
   .
   .
   .
    listenSocket.BeginAccept(
      new AsyncCallback(Server.acceptCallback),
      listenSocket);
   .
   .
   .
    // EndAccept can be called here
   .
   .
   .
  }

  public static void
    acceptCallback(IAsyncResult asyncAccept)
  {
    Socket listenSocket =
      (Socket)asyncAccept.AsyncState;

    Socket serverSocket =
      listenSocket.EndAccept(asyncAccept);

    serverSocket.BeginReceive(...);
   .
   .
   .
  }
}
```

Socket.BeginConnect(System.Net.EndPoint, System.AsyncCallback, System.Object) Method

[ILASM]

```
.method public hidebysig instance class System.IAsyncResult
BeginConnect(class System.Net.EndPoint remoteEP, class
System.AsyncCallback callback, object state)
```

[C#]

```
public IAsyncResult BeginConnect(EndPoint remoteEP,
AsyncCallback callback, object state)
```

Summary

Begins an asynchronous operation to associate the current instance with a remote endpoint.

Parameters

Parameter	Description
remoteEP	The **System.Net.EndPoint** associated with the socket to connect to.
callback	A **System.AsyncCallback** delegate, or **null**.
state	An application-defined object, or **null**.

Return Value

A **System.IAsyncResult** instance that contains information about the asynchronous operation.

Description

To release resources allocated by the **System.Net.Sockets.Socket.BeginConnect** method, call the **System.Net.Sockets.Socket.EndConnect** method, and specify the **System.IAsyncResult** object returned by this method.

[*Note:* The **System.Net.Sockets.Socket.EndConnect** method should be called exactly once for each call to the **System.Net.Sockets.Socket.BeginConnect** method.]

If the *callback* parameter is not **null**, the method referenced by *callback* is invoked when the asynchronous operation completes. The **System.IAsyncResult** object returned by this method is passed as the argument to the method referenced by *callback*. The method referenced by *callback* can retrieve the results of the operation by calling the **System.Net.Sockets.Socket.EndConnect** method.

The *state* parameter can be any object that the caller wishes to have available for the duration of the asynchronous operation. This object is available via the **System.IAsyncResult.AsyncState** property of the object returned by this method.

To determine the connection status, check the **System.Net.Sockets.Socket.Connected** property, or use either the **System.Net.Sockets.Socket.Poll** or **System.Net.Sockets.Socket.Select** method.

[*Note:* For more information, see **System.Net.Sockets.Socket.Connect**, the synchronous version of this method.]

Exceptions

Exception	Condition
System.ArgumentNullException	*remoteEP* is **null**.
System.Net.Sockets.SocketException	An error occurred while starting the operation. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.
System.Security.SecurityException	A caller higher in the call stack does not have permission for the requested operation.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, which uses the **System.Net.Sockets.Socket.BeginConnect** method, see the **System.Net.Sockets.Socket** class overview.

Permissions

Permission	Description

System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by *remoteEP*. [*Note:* See **System.Net.NetworkAccess.Connect**.]

Socket.BeginReceive(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags, System.AsyncCallback, System.Object) Method

```
[ILASM]
.method public hidebysig instance class System.IAsyncResult
BeginReceive(class System.Byte[] buffer, int32 offset,
int32 size, valuetype System.Net.Sockets.SocketFlags
socketFlags, class System.AsyncCallback callback, object
state)
```

```
[C#]
public IAsyncResult BeginReceive(byte[] buffer, int offset,
int size, SocketFlags socketFlags, AsyncCallback callback,
object state)
```

Summary

Begins an asynchronous operation to receive data from a socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
offset	A **System.Int32** containing the zero-based position in *buffer* to begin storing the received data.
size	A **System.Int32** containing the number of bytes to receive.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.
callback	A **System.AsyncCallback** delegate, or **null**.
state	An application-defined object, or **null**.

Return Value

A **System.IAsyncResult** instance that contains information about the asynchronous operation.

Description

To retrieve the results of the operation and release resources allocated by the **System.Net.Sockets.Socket.BeginReceive** method, call the **System.Net.Sockets.Socket.EndReceive** method, and specify the **System.IAsyncResult** object returned by this method.

[*Note:* The **System.Net.Sockets.Socket.EndReceive** method should be called exactly once for each call to the **System.Net.Sockets.Socket.BeginReceive** method.]

If the *callback* parameter is not **null**, the method referenced by *callback* is invoked when the asynchronous operation completes. The **System.IAsyncResult** object returned by this method is passed as the argument to the method referenced by *callback*. The method referenced by *callback* can retrieve the results of the operation by calling the **System.Net.Sockets.Socket.EndReceive** method.

The *state* parameter can be any object that the caller wishes to have available for the duration of the asynchronous operation. This object is available via the **System.IAsyncResult.AsyncState** property of the object returned by this method.

[*Note:* For more information, see **System.Net.Sockets.Socket.Receive**, the synchronous version of this method.]

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or- *size* < 0. -or- *size* > *buffer*.Length - *offset*.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket.

	[*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, which uses the **System.Net.Sockets.Socket.BeginReceive** method, see the **System.Net.Sockets.Socket** class overview.

Socket.BeginReceiveFrom(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags, System.Net.EndPoint&, System.AsyncCallback, System.Object) Method

[ILASM]

```
.method public hidebysig instance class System.IAsyncResult
BeginReceiveFrom(class System.Byte[] buffer, int32 offset,
int32 size, valuetype System.Net.Sockets.SocketFlags
socketFlags, class System.Net.EndPoint& remoteEP, class
System.AsyncCallback callback, object state)
```

[C#]

```
public IAsyncResult BeginReceiveFrom(byte[] buffer, int
offset, int size, SocketFlags socketFlags, ref EndPoint
remoteEP, AsyncCallback callback, object state)
```

Summary

Begins an asynchronous operation to receive data from a socket and, for connectionless protocols, store the endpoint associated with the socket that sent the data.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
offset	A **System.Int32** containing the zero-based position in *buffer* to begin storing the received data.
size	A **System.Int32** containing the number of bytes to receive.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.
remoteEP	An instance of a class derived from the **System.Net.EndPoint** class, which contains the endpoint associated with the socket that sent the data.
callback	A **System.AsyncCallback** delegate, or **null**.
state	An application-defined object, or **null**.

Return Value

A **System.IAsyncResult** instance that contains information about the asynchronous operation.

Description

To retrieve the results of the operation and release resources allocated by the **System.Net.Sockets.Socket.BeginReceiveFrom** method, call the **System.Net.Sockets.Socket.EndReceiveFrom** method, and specify the **System.IAsyncResult** object returned by this method.

[*Note:* The **System.Net.Sockets.Socket.EndReceiveFrom** method should be called exactly once for each call to the **System.Net.Sockets.Socket.BeginReceiveFrom** method.]

If the *callback* parameter is not **null**, the method referenced by *callback* is invoked when the asynchronous operation completes. The **System.IAsyncResult** object returned by this method is passed as the argument to the method referenced by *callback*. The method referenced by *callback* can retrieve the results of the operation by calling the **System.Net.Sockets.Socket.EndReceiveFrom** method.

The *state* parameter can be any object that the caller wishes to have available for the duration of the asynchronous operation. This object is available via the **System.IAsyncResult.AsyncState** property of the object returned by this method.

[*Note:* For more information, see **System.Net.Sockets.Socket.ReceiveFrom**, the synchronous version of this method.]

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* is **null**. -or- *remoteEP* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or-

	size < 0. -or- *size* > *buffer*.Length - *offset*.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.
System.Security.SecurityException	A caller in the call stack does not have the required permissions.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, see **System.Net.Sockets.Socket**.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept a connection on the endpoint defined by the **System.Net.Sockets.Socket.LocalEndPoint** property of the current instance. See **System.Net.NetworkAccess.Accept**. Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.BeginSend(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags, System.AsyncCallback, System.Object) Method

[ILASM]

```
.method public hidebysig instance class System.IAsyncResult
BeginSend(class System.Byte[] buffer, int32 offset, int32
size, valuetype System.Net.Sockets.SocketFlags socketFlags,
class System.AsyncCallback callback, object state)
```

[C#]

```
public IAsyncResult BeginSend(byte[] buffer, int offset,
int size, SocketFlags socketFlags, AsyncCallback callback,
object state)
```

Summary

Begins an asynchronous operation to send data to a connected socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array storing data to send to the socket.
offset	A **System.Int32** containing the zero-based position in *buffer* containing the starting location of the data to send.
size	A **System.Int32** containing the number of bytes to send.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.
callback	A **System.AsyncCallback** delegate, or **null**.
state	An application-defined object, or **null**.

Return Value

A **System.IAsyncResult** instance that contains information about the asynchronous operation.

Description

To retrieve the results of the operation and release resources allocated by the **System.Net.Sockets.Socket.BeginSend** method, call the **System.Net.Sockets.Socket.EndSend** method, and specify the **System.IAsyncResult** object returned by this method.

[*Note:* The **System.Net.Sockets.Socket.EndSend** method should be called exactly once for each call to the **System.Net.Sockets.Socket.BeginSend** method.]

If the *callback* parameter is not **null**, the method referenced by *callback* is invoked when the asynchronous operation completes. The **System.IAsyncResult** object returned by this method is passed as the argument to the method referenced by *callback*. The method referenced by *callback* can retrieve the results of the operation by calling the **System.Net.Sockets.Socket.EndSend** method.

The *state* parameter can be any object that the caller wishes to have available for the duration of the asynchronous operation. This object is available via the **System.IAsyncResult.AsyncState** property of the object returned by this method.

[*Note:* For more information, see **System.Net.Sockets.Socket.Send**, the synchronous version of this method.]

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or- *size* < 0. -or- *size* > *buffer*.Length - *offset*.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket.

	[*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, which uses the **System.Net.Sockets.Socket.BeginSend** method, see the **System.Net.Sockets.Socket** class overview.

Socket.BeginSendTo(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags, System.Net.EndPoint, System.AsyncCallback, System.Object) Method

[ILASM]

```
.method public hidebysig instance class System.IAsyncResult
BeginSendTo(class System.Byte[] buffer, int32 offset, int32
size, valuetype System.Net.Sockets.SocketFlags socketFlags,
class System.Net.EndPoint remoteEP, class
System.AsyncCallback callback, object state)
```

[C#]

```
public IAsyncResult BeginSendTo(byte[] buffer, int offset,
int size, SocketFlags socketFlags, EndPoint remoteEP,
AsyncCallback callback, object state)
```

Summary

Begins an asynchronous operation to send data to the socket associated with the specified endpoint.

Parameters

Parameter	Description
buffer	A **System.Byte** array storing data to send to the socket.
offset	A **System.Int32** containing the zero-based position in *buffer* to begin sending data.
size	A **System.Int32** containing the number of bytes to send.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.
remoteEP	The **System.Net.EndPoint** associated with the socket to receive the data.
callback	A **System.AsyncCallback** delegate, or **null**.
state	An application-defined object, or **null**.

Return Value

A **System.IAsyncResult** instance that contains information about the asynchronous operation.

Description

To retrieve the results of the operation and release resources allocated by the **System.Net.Sockets.Socket.BeginSendTo** method, call the **System.Net.Sockets.Socket.EndSendTo** method, and specify the **System.IAsyncResult** object returned by this method.

[*Note:* The **System.Net.Sockets.Socket.EndSendTo** method should be called exactly once for each call to the **System.Net.Sockets.Socket.BeginSendTo** method.]

If the *callback* parameter is not **null**, the method referenced by *callback* is invoked when the asynchronous operation completes. The **System.IAsyncResult** object returned by this method is passed as the argument to the method referenced by *callback*. The method referenced by *callback* can retrieve the results of the operation by calling the **System.Net.Sockets.Socket.EndSendTo** method.

The *state* parameter can be any object that the caller wishes to have available for the duration of the asynchronous operation. This object is available via the **System.IAsyncResult.AsyncState** property of the object returned by this method.

[*Note:* For more information, see **System.Net.Sockets.Socket.SendTo**, the synchronous version of this method.]

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* is **null**. -or- *remoteEP* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or- *size* < 0. -or-

	size > *buffer*.Length - *offset*.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.
System.Security.SecurityException	A caller in the call stack does not have the required permissions.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, see the **System.Net.Sockets.Socket** class overview.

Permissions

Permission	**Description**
System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.Bind(System.Net.EndPoint) Method

[ILASM]

```
.method public hidebysig instance void Bind(class
System.Net.EndPoint localEP)
```

[C#]

```
public void Bind(EndPoint localEP)
```

Summary

Associates the current instance with a local endpoint.

Parameters

Parameter	Description
localEP	The local **System.Net.EndPoint** to be associated with the socket.

Description

This method sets the **System.Net.Sockets.Socket.LocalEndPoint** property of the current instance to *localEP*.

[*Note:* For connection-oriented protocols, this method is generally used only on the server-side and is required to be called before the first call to the **System.Net.Sockets.Socket.Listen** method. On the client-side, binding is usually performed implicitly by the **System.Net.Sockets.Socket.Connect** method.

For connectionless protocols, the **System.Net.Sockets.Socket.Connect** **System.Net.Sockets.Socket.SendTo**, and **System.Net.Sockets.Socket.BeginSendTo** methods bind the current instance to the local endpoint if the current instance has not previously been bound.]

Exceptions

Exception	Condition
System.ArgumentNullException	*localEP* is **null**.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see

	the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.
System.Security.SecurityException	A caller in the call stack does not have the required permission.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept connections on the endpoint defined by *localEP*. See **System.Net.NetworkAccess.Accept**.

Socket.Close() Method

[ILASM]
```
.method public hidebysig instance void Close()
```

[C#]
```
public void Close()
```

Summary

Closes the current instance and releases all managed and unmanaged resources allocated by the current instance.

Description

This method calls the **System.Net.Sockets.Socket.Dispose**(**System.Boolean**) method with the argument set to **true**, which frees both managed and unmanaged resources used by the current instance.

The socket attempts to perform a graceful closure when the **System.Net.Sockets.SocketOptionName.Linger** socket option is enabled and set to a non-zero linger time. In all other cases, closure is forced and any pending data is lost.

Socket.Connect(System.Net.EndPoint) Method

[ILASM]
```
.method public hidebysig instance void Connect(class
System.Net.EndPoint remoteEP)
```

[C#]
```
public void Connect(EndPoint remoteEP)
```

Summary

Associates the current instance with a remote endpoint.

Parameters

Parameter	Description
remoteEP	The **System.Net.EndPoint** associated with the socket to connect to.

Description

This method sets the **System.Net.Sockets.Socket.RemoteEndPoint** property of the current instance to *remoteEP*.

[*Note:* For connection-oriented protocols, this method establishes a connection between the current instance and the socket associated with *remoteEP*. This method is used only on the client-side. The **System.Net.Sockets.Socket.Accept** method establishes the connection on the server-side. Once the connection has been made, data can be sent using the **System.Net.Sockets.Socket.Send** method, and received using the **System.Net.Sockets.Socket.Receive** method.

For connectionless protocols, the **System.Net.Sockets.Socket.Connect** method can be used from both client and server-sides, allowing the use of the **System.Net.Sockets.Socket.Send** method instead of the **System.Net.Sockets.Socket.SendTo** method. The **System.Net.Sockets.Socket.RemoteEndPoint** property is set to *remoteEP* and the **System.Net.Sockets.Socket.LocalEndPoint** property is set to a value determined by the protocol; however, a connection is not established. Subsequent data is required to be received on the endpoint set in the **System.Net.Sockets.Socket.LocalEndPoint** property.]

Exceptions

Exception	Condition
System.ArgumentNullException	*remoteEP* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.
System.Security.SecurityException	A caller in the call stack does not have the required permission.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.Dispose(System.Boolean) Method

```
[ILASM]
.method family hidebysig virtual void Dispose(bool
disposing)

[C#]
protected virtual void Dispose(bool disposing)
```

Summary

Closes the current instance, releases the unmanaged resources allocated by the current instance, and optionally releases the managed resources.

Parameters

Parameter	Description
disposing	A **System.Boolean**. Specify **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Behaviors

This method closes the current **System.Net.Sockets.Socket** instance and releases all unmanaged resources allocated by the current instance. When *disposing* is **true**, this method also releases all resources held by any managed objects allocated by the current instance.

Default

This method closes the current **System.Net.Sockets.Socket** instance but does not release any managed resources.

How and When to Override

The **System.Net.Sockets.Socket.Dispose** method can be called multiple times by other objects. When overriding this method, do not reference objects that have been previously disposed in an earlier call.

Usage

Use this method to release resources allocated by the current instance.

Socket.EndAccept(System.IAsyncResult) Method

```
[ILASM]
.method public hidebysig instance class
System.Net.Sockets.Socket EndAccept(class
System.IAsyncResult asyncResult)

[C#]
public Socket EndAccept(IAsyncResult asyncResult)
```

Summary

Ends an asynchronous call to accept an incoming connection request.

Parameters

Parameter	Description
asyncResult	A **System.IAsyncResult** object that holds the state information for the asynchronous operation.

Return Value

A new connected **System.Net.Sockets.Socket** instance.

Description

This method blocks if the asynchronous operation has not completed.

The **System.Net.Sockets.Socket.EndAccept** method completes an asynchronous request that was started with a call to the **System.Net.Sockets.Socket.BeginAccept** method. The object specified for the *asyncResult* parameter is required to be the same object as was returned by the **System.Net.Sockets.Socket.BeginAccept** method call that began the request.

If the **System.Net.Sockets.Socket.EndAccept** method is invoked via the **System.AsyncCallback** delegate specified to the **System.Net.Sockets.Socket.BeginAccept** method, the *asyncResult* parameter is the **System.IAsyncResult** argument passed to the delegate's method.

Exceptions

Exception	Condition

System.ArgumentNullException	*asyncResult* is **null**.
System.ArgumentException	*asyncResult* was not returned by the current instance from a call to the **System.Net.Sockets.Socket.BeginAccept** method.
System.InvalidOperationException	**System.Net.Sockets.Socket.EndAccept** was previously called for this operation.
System.Net.Sockets.SocketException	An error occurred during the operation. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, which uses the **System.Net.Sockets.Socket.EndAccept** method, see the **System.Net.Sockets.Socket** class overview.

Socket.EndConnect(System.IAsyncResult) Method

```
[ILASM]
.method public hidebysig instance void EndConnect(class
System.IAsyncResult asyncResult)

[C#]
public void EndConnect(IAsyncResult asyncResult)
```

Summary

Ends an asynchronous call to associate the current instance with a remote endpoint.

Parameters

Parameter	Description
asyncResult	A **System.IAsyncResult** object that holds the state information for the asynchronous operation.

Description

This method blocks if the asynchronous operation has not completed.

The **System.Net.Sockets.Socket.EndConnect** method completes an asynchronous request that was started with a call to the **System.Net.Sockets.Socket.BeginConnect** method. The object specified for the *asyncResult* parameter is required to be the same object as was returned by the **System.Net.Sockets.Socket.BeginConnect** method call that began the request.

If the **System.Net.Sockets.Socket.EndConnect** method is invoked via the **System.AsyncCallback** delegate specified to the **System.Net.Sockets.Socket.BeginConnect** method, the *asyncResult* parameter is the **System.IAsyncResult** argument passed to the delegate's method.

Exceptions

Exception	Condition
System.ArgumentNullException	*asyncResult* is **null**.
System.ArgumentException	*asyncResult* was not returned by the current instance from a call to the **System.Net.Sockets.Socket.BeginConnect**

	method.
System.InvalidOperationException	**System.Net.Sockets.Socket.EndConnect** was previously called for this operation.
System.Net.Sockets.SocketException	An error occurred during the operation. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, which uses the **System.Net.Sockets.Socket.EndConnect** method, see the **System.Net.Sockets.Socket** class overview.

Socket.EndReceive(System.IAsyncResult) Method

```
[ILASM]
.method public hidebysig instance int32 EndReceive(class
System.IAsyncResult asyncResult)

[C#]
public int EndReceive(IAsyncResult asyncResult)
```

Summary

Ends an asynchronous call to receive data from a socket.

Parameters

Parameter	Description
asyncResult	A **System.IAsyncResult** object that holds the state information for the asynchronous operation.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method blocks if the asynchronous operation has not completed.

The **System.Net.Sockets.Socket.EndReceive** method completes an asynchronous request that was started with a call to the **System.Net.Sockets.Socket.BeginReceive** method. The object specified for the *asyncResult* parameter is required to be the same object as was returned by the **System.Net.Sockets.Socket.BeginReceive** method call that began the request.

If the **System.Net.Sockets.Socket.EndReceive** method is invoked via the **System.AsyncCallback** delegate specified to the **System.Net.Sockets.Socket.BeginReceive** method, the *asyncResult* parameter is the **System.IAsyncResult** argument passed to the delegate's method.

Exceptions

Exception	Condition

System.ArgumentNullException	*asyncResult* is **null**.
System.ArgumentException	*asyncResult* was not returned by the current instance from a call to the **System.Net.Sockets.Socket.BeginReceive** method.
System.InvalidOperationException	**System.Net.Sockets.Socket.EndReceive** was previously called for this operation.
System.Net.Sockets.SocketException	An error occurred during the operation. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, which uses the **System.Net.Sockets.Socket.EndReceive** method, see the **System.Net.Sockets.Socket** class overview.

Socket.EndReceiveFrom(System.IAsyncResult, System.Net.EndPoint&) Method

[ILASM]

```
.method public hidebysig instance int32
EndReceiveFrom(class System.IAsyncResult asyncResult, class
System.Net.EndPoint& endPoint)
```

[C#]

```
public int EndReceiveFrom(IAsyncResult asyncResult, ref
EndPoint endPoint)
```

Summary

Ends an asynchronous call to receive data from a socket and store the endpoint associated with the socket that sent the data.

Parameters

Parameter	Description
asyncResult	A **System.IAsyncResult** object that holds the state information for the asynchronous operation.
endPoint	A reference to the **System.Net.EndPoint** associated with the socket that sent the data.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method blocks if the asynchronous operation has not completed.

The **System.Net.Sockets.Socket.EndReceiveFrom** method completes an asynchronous request that was started with a call to the **System.Net.Sockets.Socket.BeginReceiveFrom** method. The object specified for the *asyncResult* parameter is required to be the same object as was returned by the **System.Net.Sockets.Socket.BeginReceiveFrom** method call that began the request.

If the **System.Net.Sockets.Socket.EndReceiveFrom** method is invoked via the **System.AsyncCallback** delegate specified to the **System.Net.Sockets.Socket.BeginReceiveFrom** method, the *asyncResult* parameter is the **System.IAsyncResult** argument passed to the delegate's method.

Exceptions

Exception	Condition
System.ArgumentNullException	*asyncResult* is **null**.
System.ArgumentException	*asyncResult* was not returned by the current instance from a call to the **System.Net.Sockets.Socket.BeginReceiveFro** method.
System.InvalidOperationException	**System.Net.Sockets.Socket.EndReceiveFrom** was previously called for this operation.
System.Net.Sockets.SocketException	An error occurred during the operation. [*Note:* Fo additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, see the **System.Net.Sockets.Socket** class overview.

Socket.EndSend(System.IAsyncResult) Method

```
[ILASM]
.method public hidebysig instance int32 EndSend(class
System.IAsyncResult asyncResult)

[C#]
public int EndSend(IAsyncResult asyncResult)
```

Summary

Ends an asynchronous call to send data to a connected socket.

Parameters

Parameter	Description
asyncResult	A **System.IAsyncResult** object that holds the state information for the asynchronous operation.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method blocks if the asynchronous operation has not completed.

The **System.Net.Sockets.Socket.EndSend** method completes an asynchronous request that was started with a call to the **System.Net.Sockets.Socket.BeginSend** method. The object specified for the *asyncResult* parameter is required to be the same object as was returned by the **System.Net.Sockets.Socket.BeginSend** method call that began the request.

If the **System.Net.Sockets.Socket.EndSend** method is invoked via the **System.AsyncCallback** delegate specified to the **System.Net.Sockets.Socket.BeginSend** method, the *asyncResult* parameter is the **System.IAsyncResult** argument passed to the delegate's method.

Exceptions

Exception	Condition

System.ArgumentNullException	*asyncResult* is **null**.
System.ArgumentException	*asyncResult* was not returned by the current instance from a call to the **System.Net.Sockets.Socket.BeginSend** method.
System.InvalidOperationException	**System.Net.Sockets.Socket.EndSend** was previously called for this operation.
System.Net.Sockets.SocketException	An error occurred during the operation. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, which uses the **System.Net.Sockets.Socket.EndSend** method, see the **System.Net.Sockets.Socket** class overview.

Socket.EndSendTo(System.IAsyncResult) Method

[ILASM]

```
.method public hidebysig instance int32 EndSendTo(class
System.IAsyncResult asyncResult)
```

[C#]

```
public int EndSendTo(IAsyncResult asyncResult)
```

Summary

Ends an asynchronous call to send data to a socket associated with a specified endpoint.

Parameters

Parameter	Description
asyncResult	A **System.IAsyncResult** object that holds the state information for the asynchronous operation.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method blocks if the asynchronous operation has not completed.

The **System.Net.Sockets.Socket.EndSendTo** method completes an asynchronous request that was started with a call to the **System.Net.Sockets.Socket.BeginSendTo** method. The object specified for the *asyncResult* parameter is required to be the same object as was returned by the **System.Net.Sockets.Socket.BeginSendTo** method call that began the request.

If the **System.Net.Sockets.Socket.EndSendTo** method is invoked via the **System.AsyncCallback** delegate specified to the **System.Net.Sockets.Socket.BeginSendTo** method, the *asyncResult* parameter is the **System.IAsyncResult** argument passed to the delegate's method.

Exceptions

Exception	**Condition**
System.ArgumentNullException	*asyncResult* is **null**.
System.ArgumentException	*asyncResult* was not returned by the current instance from a call to the **System.Net.Sockets.Socket.SendTo** method.
System.InvalidOperationException	**System.Net.Sockets.Socket.EndSendTo** was previously called for this operation.
System.Net.Sockets.SocketException	An error occurred during the operation. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

For an outline of an asynchronous operation, see the **System.Net.Sockets.Socket.BeginAccept** method. For the complete example, see the **System.Net.Sockets.Socket** class overview.

Socket.Finalize() Method

[ILASM]
```
.method family hidebysig virtual void Finalize()
```

[C#]
```
~Socket()
```

Summary

Closes the current instance and releases unmanaged resources allocated by the current instance.

Description

[*Note:* Application code does not call this method; it is automatically invoked during garbage collection unless finalization by the garbage collector has been disabled. For more information, see **System.GC.SuppressFinalize**, and **System.Object.Finalize**.

This method calls **System.Net.Sockets.NetworkStream.Dispose**(**false**) to free unmanaged resources used by the current instance.

This method overrides **System.Object.Finalize**.]

Socket.GetHashCode() Method

[ILASM]

```
.method public hidebysig virtual int32 GetHashCode()
```

[C#]

```
public override int GetHashCode()
```

Summary

Generates a hash code for the current instance.

Return Value

A **System.Int32** containing the hash code for the current instance.

Description

The algorithm used to generate the hash code is unspecified.

[*Note:* This method overrides **System.Object.GetHashCode**.]

Socket.GetSocketOption(System.Net.Sockets.SocketOptionLevel, System.Net.Sockets.SocketOptionName) Method

```
[ILASM]
.method public hidebysig instance object
GetSocketOption(valuetype
System.Net.Sockets.SocketOptionLevel optionLevel, valuetype
System.Net.Sockets.SocketOptionName optionName)

[C#]
public object GetSocketOption(SocketOptionLevel
optionLevel, SocketOptionName optionName)
```

Summary

Retrieves an object containing the value of the specified socket option.

Parameters

Parameter	Description
optionLevel	One of the values defined in the **System.Net.Sockets.SocketOptionLevel** enumeration.
optionName	One of the values defined in the **System.Net.Sockets.SocketOptionName** enumeration.

Return Value

The following table describes the values returned by this method.

optionName	Return value
Linger	An instance of the **System.Net.Sockets.LingerOption** class.
AddMembership -or- **DropMembership**	An instance of the **System.Net.Sockets.MulticastOption** class.
All other values defined in the **System.Net.Sockets.SocketOptionName** enumeration.	A **System.Int32** containing the value of the option.

Description

Socket options determine the behavior of the current instance.

optionLevel and *optionName* are not independent. See the **System.Net.Sockets.Socket.SetSocketOption**(**SocketOptionLevel**, **SocketOptionName**, **Int32**) method for a listing of the values of the **System.Net.Sockets.SocketOptionName** enumeration grouped by **System.Net.Sockets.SocketOptionLevel**.

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

The following example gets the state of the linger option and the size of the receive buffer, changes the values of both, then gets the new values.

[C#]

```
using System;
using System.Net.Sockets;

class OptionTest{

  public static void Main() {

    // Get the current option values.
    Socket someSocket =
      new Socket(AddressFamily.InterNetwork,
                 SocketType.Stream,
                 ProtocolType.Tcp);

    LingerOption lingerOp =
      (LingerOption)someSocket.GetSocketOption(
                    SocketOptionLevel.Socket,
                    SocketOptionName.Linger);

    int receiveBuffer =
      (int)someSocket.GetSocketOption(
           SocketOptionLevel.Socket,
           SocketOptionName.ReceiveBuffer);

    Console.WriteLine(
      "Linger option is {0} and set to {1} seconds.",
      lingerOp.Enabled.ToString(),
      lingerOp.LingerTime.ToString());

    Console.WriteLine(
      "Size of the receive buffer is {0} bytes.",
      receiveBuffer.ToString());

    // Change the options.
    lingerOp = new LingerOption(true, 10);
    someSocket.SetSocketOption(
      SocketOptionLevel.Socket,
      SocketOptionName.Linger,
      lingerOp);

    someSocket.SetSocketOption(
      SocketOptionLevel.Socket,
      SocketOptionName.ReceiveBuffer,
      2048);

    Console.WriteLine(
      "The SetSocketOption method has been called.");

    // Get the new option values.
    lingerOp =
      (LingerOption)someSocket.GetSocketOption(
                    SocketOptionLevel.Socket,
                    SocketOptionName.Linger);

    receiveBuffer =
      (int)someSocket.GetSocketOption(
           SocketOptionLevel.Socket,
           SocketOptionName.ReceiveBuffer);

    Console.WriteLine(
      "Linger option is now {0} and set to {1} seconds.",
      lingerOp.Enabled.ToString(),
      lingerOp.LingerTime.ToString());

    Console.WriteLine(
      "Size of the receive buffer is now {0} bytes.",
      receiveBuffer.ToString());
  }
}
```

The output is

```
Linger option is False and set to 0 seconds.


Size of the receive buffer is 8192 bytes.


The SetSocketOption method has been called.


Linger option is now True and set to 10 seconds.


Size of the receive buffer is now 2048 bytes.
```

Socket.GetSocketOption(System.Net.Sockets.SocketOptionLevel, System.Net.Sockets.SocketOptionName, System.Byte[]) Method

[ILASM]

```
.method public hidebysig instance void
GetSocketOption(valuetype
System.Net.Sockets.SocketOptionLevel optionLevel, valuetype
System.Net.Sockets.SocketOptionName optionName, class
System.Byte[] optionValue)
```

[C#]

```
public void GetSocketOption(SocketOptionLevel optionLevel,
SocketOptionName optionName, byte[] optionValue)
```

Summary

Retrieves the value of the specified socket option.

Parameters

Parameter	Description
optionLevel	One of the values defined in the **System.Net.Sockets.SocketOptionLevel** enumeration.
optionName	One of the values defined in the **System.Net.Sockets.SocketOptionName** enumeration.
optionValue	A **System.Byte** array that receives the value of the specified socket option.

Description

Socket options determine the behavior of the current instance.

Upon successful completion, the array specified by the *optionValue* parameter contains the value of the specified socket option.

When the length of the *optionValue* array is smaller than the number of bytes required to store the value of the specified socket option, a **System.Net.Sockets.SocketException** exception is thrown.

Exceptions

Exception	Condition

System.Net.Sockets.SocketException	*optionValue* is too small to store the value of the specified socket option. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.GetSocketOption(System.Net.Sockets.SocketOptionLevel, System.Net.Sockets.SocketOptionName, System.Int32) Method

```
[ILASM]
.method public hidebysig instance class System.Byte[]
GetSocketOption(valuetype
System.Net.Sockets.SocketOptionLevel optionLevel, valuetype
System.Net.Sockets.SocketOptionName optionName, int32
optionLength)

[C#]
public byte[] GetSocketOption(SocketOptionLevel
optionLevel, SocketOptionName optionName, int optionLength)
```

Summary

Retrieves the value of the specified socket option.

Parameters

Parameter	Description
optionLevel	One of the values defined in the **System.Net.Sockets.SocketOptionLevel** enumeration.
optionName	One of the values defined in the **System.Net.Sockets.SocketOptionName** enumeration.
optionLength	A **System.Int32** containing the maximum length, in bytes, of the value of the specified socket option.

Return Value

A **System.Byte** array containing the value of the specified socket option.

Description

Socket options determine the behavior of the current instance.

The *optionLength* parameter is used to allocate an array to store the value of the specified option. When this value is smaller than the number of bytes required to store the value of the specified option, a **System.Net.Sockets.SocketException** exception is thrown. When this value is greater than or equal to the number of bytes required to store the value of the specified option, the array returned by this method is allocated to be exactly the required length.

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	*optionLength* is smaller than the number of bytes required to store the value of the specified socket option. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.IOControl(System.Int32, System.Byte[], System.Byte[]) Method

[ILASM]

```
.method public hidebysig instance int32 IOControl(int32
ioControlCode, class System.Byte[] optionInValue, class
System.Byte[] optionOutValue)
```

[C#]

```
public int IOControl(int ioControlCode, byte[]
optionInValue, byte[] optionOutValue)
```

Summary

Provides low-level access to the socket, the transport protocol, or the communications subsystem.

Parameters

Parameter	Description
ioControlCode	A **System.Int32** containing the control code of the operation to perform.
optionInValue	A **System.Byte** array containing the input data required by the operation.
optionOutValue	A **System.Byte** array containing the output data supplied by the operation.

Return Value

A **System.Int32** containing the length of the *optionOutValue* array after the method returns.

Description

If an attempt is made to change the blocking mode of the current instance, an exception is thrown. Use the **System.Net.Sockets.Socket.Blocking** property to change the blocking mode.

The control codes and their requirements are implementation defined. Do not use this method if platform independence is a requirement.

[*Note:* Input data is not required for all control codes. Output data is not supplied by all control codes and, if not supplied, the return value is 0.]

Exceptions

Exception	Condition
System.InvalidOperationException	An attempt was made to change the blocking mode. [*Note:* Use the **System.Net.Sockets.Socket.Blocking** property to change the blocking mode.]
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Example

The following example gets the number of bytes of available data to be read and writes the result to the console on a Windows system. The remote endpoint (remoteEndpoint) to connect to may need to be changed to a value that is valid on the current system.

[C#]

```
using System;
using System.Net;
using System.Net.Sockets;

class App {

  static void Main() {

    IPAddress remoteAddress =
    Dns.Resolve(Dns.GetHostName()).AddressList[0];

    IPEndPoint remoteEndpoint =
      new IPEndPoint(remoteAddress, 80);

    Socket someSocket =
      new Socket(AddressFamily.InterNetwork,
                 SocketType.Stream,
                 ProtocolType.Tcp);

    someSocket.Connect(remoteEndpoint);

    int fionRead = 0x4004667F;
    byte[]inValue = {0x00, 0x00, 0x00, 0x00};
    byte[]outValue = {0x00, 0x00, 0x00, 0x00};

    someSocket.IOControl(fionRead, inValue, outValue);

    uint bytesAvail = BitConverter.ToUInt32(outValue, 0);

    Console.WriteLine(
      "There are {0} bytes available to be read.",
      bytesAvail.ToString());
  }
}
```

The output is

```
There are 0 bytes available to be read.
```

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	Requires permission to access unmanaged code. See **System.Security.Permissions.SecurityPermissionFlag** **UnmanagedCode**.

Socket.Listen(System.Int32) Method

```
[ILASM]
.method public hidebysig instance void Listen(int32
backlog)

[C#]
public void Listen(int backlog)
```

Summary

Places the current instance into the listening state where it waits for incoming connection requests.

Parameters

Parameter	Description
backlog	A **System.Int32** containing the maximum length of the queue of pending connections.

Description

Once this method is called, incoming connection requests are placed in a queue. The maximum size of the queue is specified by the *backlog* parameter. The size of the queue is limited to legal values by the underlying protocol. Illegal values of the *backlog* parameter are replaced with a legal value, which is implementation defined.

If a connection request arrives and the queue is full, a **System.Net.Sockets.SocketException** is thrown on the client.

A socket in the listening state has no remote endpoint associated with it. Attempting to access the **System.Net.Sockets.Socket.RemoteEndPoint** property throws a **System.Net.Sockets.SocketException** exception.

This method is ignored if called more than once on the current instance.

[*Note:* This method is used only on the server-side of connection-oriented protocols. Call the **System.Net.Sockets.Socket.Bind** method before this method is called the first time. Call the **System.Net.Sockets.Socket.Listen** method before the first call to the **System.Net.Sockets.Socket.Accept** method.]

Exceptions

Exception	**Condition**
System.Net.Sockets.SocketException	The **System.Net.Sockets.Socket.Connected** property of the current instance is true, or an error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.Poll(System.Int32, System.Net.Sockets.SelectMode) Method

[ILASM]
```
.method public hidebysig instance bool Poll(int32
microSeconds, valuetype System.Net.Sockets.SelectMode mode)
```

[C#]
```
public bool Poll(int microSeconds, SelectMode mode)
```

Summary

Determines the read, write, or error status of the current instance.

Parameters

Parameter	Description
microSeconds	A **System.Int32** containing the time to wait for a response, in microseconds. Set the *microSeconds* parameter to a negative value to wait indefinitely for a response.
mode	One of the values defined in the **System.Net.Sockets.SelectMode** enumeration.

Return Value

A **System.Boolean** where **true** indicates the current instance satisfies at least one of the conditions in the following table corresponding to the specified **System.Net.Sockets.SelectMode** value; otherwise, **false**. **false** is returned if the status of the current instance cannot be determined within the time specified by *microSeconds*.

SelectMode value	Condition
SelectRead	Data is available for reading (includes out-of-band data if the **System.Net.Sockets.SocketOptionName.OutOfBandInline** value defined in the **System.Net.Sockets.SocketOptionName** enumeration is set). -or- The socket is in the listening state with a pending connection, and the **System.Net.Sockets.Socket.Accept** method has been called and is guaranteed to succeed without blocking. -or-

	The connection has been closed, reset, or terminated.
SelectWrite	Data can be sent. -or- A non-blocking **System.Net.Sockets.Socket.Connect** method is being processed and the connection has succeeded.
SelectError	The **System.Net.Sockets.SocketOptionName.OutOfBandInline** value defined in the **System.Net.Sockets.SocketOptionName** enumeration is not set and out-of-band data is available. -or- A non-blocking **System.Net.Sockets.Socket.Connect** method is being processed and the connection has failed.

Exceptions

Exception	Condition
System.NotSupportedException	*mode* is not one of the values defined in the **System.Net.Sockets.SelectMode** enumeration.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.Receive(System.Byte[], System.Int32, System.Net.Sockets.SocketFlags) Method

```
[ILASM]
.method public hidebysig instance int32 Receive(class
System.Byte[] buffer, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags)

[C#]
public int Receive(byte[] buffer, int size, SocketFlags
socketFlags)
```

Summary

Receives data from a socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
size	A **System.Int32** containing the number of bytes to receive.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method is equivalent to **System.Net.Sockets.Socket.Receive**(*buffer*, 0, *size*, *socketFlags*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* is **null**.
System.ArgumentOutOfRangeException	*size* < 0.

	-or- *size* > *buffer*.Length.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept connections. See **System.Net.NetworkAccess.Accept**.

Socket.Receive(System.Byte[], System.Net.Sockets.SocketFlags) Method

```
[ILASM]
.method public hidebysig instance int32 Receive(class
System.Byte[] buffer, valuetype
System.Net.Sockets.SocketFlags socketFlags)

[C#]
public int Receive(byte[] buffer, SocketFlags socketFlags)
```

Summary

Receives data from a socket.

Parameters

Parameter	**Description**
buffer	A **System.Byte** array to store data received from the socket.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method is equivalent to **System.Net.Sockets.Socket.Receive**(*buffer*, 0, *buffer*.Length, *socketFlags*).

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of

	values.
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	**Description**
System.Net.SocketPermission	Requires permission to accept connections. [*Note:* See **System.Net.NetworkAccess.Accept**.]

Socket.Receive(System.Byte[]) Method

```
[ILASM]
.method public hidebysig instance int32 Receive(class
System.Byte[] buffer)

[C#]
public int Receive(byte[] buffer)
```

Summary

Receives data from a socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method is equivalent to **System.Net.Sockets.Socket.Receive**(*buffer*, 0, *buffer*.Length, **System.Net.Sockets.SocketFlags.None**).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the

	required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept connections. See **System.Net.NetworkAccess.Accept**.

Socket.Receive(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags) Method

```
[ILASM]
.method public hidebysig instance int32 Receive(class
System.Byte[] buffer, int32 offset, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags)

[C#]
public int Receive(byte[] buffer, int offset, int size,
SocketFlags socketFlags)
```

Summary

Receives data from a socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
offset	A **System.Int32** containing the zero-based position in *buffer* to begin storing the received data.
size	A **System.Int32** containing the number of bytes to receive.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.

Return Value

A **System.Int32** containing the number of bytes received.

Description

The **System.Net.Sockets.Socket.LocalEndPoint** property is required to be set before this method is called.

The **System.Net.Sockets.Socket.Blocking** property of the socket determines the behavior of this method when no incoming data is available. When **false**, the **System.Net.Sockets.SocketException** exception is thrown. When **true**, this method blocks and waits for data to arrive.

For **System.Net.Sockets.SocketType.Stream** socket types, if the remote socket was shut down gracefully, and all data was received, this method immediately returns zero, regardless of the blocking state.

For message-oriented sockets, if the message is larger than the size of *buffer*, the buffer is filled with the first part of the message, and the **System.Net.Sockets.SocketException** exception is thrown. For unreliable protocols, the excess data is lost; for reliable protocols, the data is retained by the service provider.

When the **System.Net.Sockets.SocketFlags.OutOfBand** flag is specified as part of the *socketFlags* parameter and the socket is configured for in-line reception of out-of-band (OOB) data (using the **System.Net.Sockets.SocketOptionName.OutOfBandInline** socket option) and OOB data is available, only OOB data is returned.

When the **System.Net.Sockets.SocketFlags.Peek** flag is specified as part of the *socketFlags* parameter, available data is copied into *buffer* but is not removed from the system buffer.

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or- *size* < 0. -or- *size* > *buffer*.Length - *offset*.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- The **System.Net.Sockets.Socket.LocalEndPoint** property was not set.

	-or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept a connection on the endpoint defined by the **System.Net.Sockets.Socket.LocalEndPoint** property of the current instance. See **System.Net.NetworkAccess.Accept**.

Socket.ReceiveFrom(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags, System.Net.EndPoint&) Method

[ILASM]

```
.method public hidebysig instance int32 ReceiveFrom(class
System.Byte[] buffer, int32 offset, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags, class
System.Net.EndPoint& remoteEP)
```

[C#]

```
public int ReceiveFrom(byte[] buffer, int offset, int size,
SocketFlags socketFlags, ref EndPoint remoteEP)
```

Summary

Receives data from a socket and, for connectionless protocols, stores the endpoint associated with the socket that sent the data.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
offset	A **System.Int32** containing the zero-based position in *buffer* to begin storing the received data.
size	A **System.Int32** containing the number of bytes to receive.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.
remoteEP	A reference to the **System.Net.EndPoint** associated with the socket that sent the data.

Return Value

A **System.Int32** containing the number of bytes received.

Description

For connectionless protocols, when this method successfully completes, *remoteEP* contains the endpoint associated with the socket that sent the data.

For connection-oriented protocols, *remoteEP* is left unchanged.

The **System.Net.Sockets.Socket.LocalEndPoint** property is required to be set before this method is called or a **System.Net.Sockets.SocketException** is thrown.

The **System.Net.Sockets.Socket.Blocking** property of the socket determines the behavior of this method when no incoming data is available. When **false**, the **System.Net.Sockets.SocketException** exception is thrown. When **true**, this method blocks and waits for data to arrive.

For **System.Net.Sockets.SocketType.Stream** socket types, if the remote socket was shut down gracefully, and all data was received, this method immediately returns zero, regardless of the blocking state.

For message-oriented sockets, if the message is larger than the size of *buffer*, the buffer is filled with the first part of the message, and the **System.Net.Sockets.SocketException** exception is thrown. For unreliable protocols, the excess data is lost; for reliable protocols, the data is retained by the service provider.

When the **System.Net.Sockets.SocketFlags.OutOfBand** flag is specified as part of the*socketFlags* parameter and the socket is configured for in-line reception of out-of-band (OOB) data (using the **System.Net.Sockets.SocketOptionName.OutOfBandInline** socket option) and OOB data is available, only OOB data is returned.

When the **System.Net.Sockets.SocketFlags.Peek** flag is specified as part of the *socketFlags* parameter, available data is copied into *buffer* but is not removed from the system buffer.

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* or *remoteEP* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or- *size* < 0. -or-

	size > *buffer*.Length - *offset*.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- The **System.Net.Sockets.Socket.LocalEndPoint** property was not set. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept a connection on the endpoint defined by the **System.Net.Sockets.Socket.LocalEndPoint** property of the current instance. See **System.Net.NetworkAccess.Accept**. Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.ReceiveFrom(System.Byte[], System.Int32, System.Net.Sockets.SocketFlags, System.Net.EndPoint&) Method

```
[ILASM]
.method public hidebysig instance int32 ReceiveFrom(class
System.Byte[] buffer, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags, class
System.Net.EndPoint& remoteEP)

[C#]
public int ReceiveFrom(byte[] buffer, int size, SocketFlags
socketFlags, ref EndPoint remoteEP)
```

Summary

Receives data from a socket and, for connectionless protocols, stores the endpoint associated with the socket that sent the data.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
size	A **System.Int32** containing the number of bytes to receive.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.
remoteEP	A reference to the **System.Net.EndPoint** associated with the socket that sent the data.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method is equivalent to **System.Net.Sockets.Socket.ReceiveFrom**(*buffer*, 0, *size*, *socketFlags*, *remoteEP*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* or *remoteEP* is **null**.
System.ArgumentOutOfRangeException	*size* < 0. -or- *size* > *buffer*.Length.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept connections from the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Accept**.

Socket.ReceiveFrom(System.Byte[], System.Net.Sockets.SocketFlags, System.Net.EndPoint&) Method

[ILASM]

```
.method public hidebysig instance int32 ReceiveFrom(class
System.Byte[] buffer, valuetype
System.Net.Sockets.SocketFlags socketFlags, class
System.Net.EndPoint& remoteEP)
```

[C#]

```
public int ReceiveFrom(byte[] buffer, SocketFlags
socketFlags, ref EndPoint remoteEP)
```

Summary

Receives data from a socket and, for connectionless protocols, stores the endpoint associated with the socket that sent the data.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.OutOfBand**, or **System.Net.Sockets.SocketFlags.Peek**.
remoteEP	A reference to the **System.Net.EndPoint** associated with the socket that sent the data.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method is equivalent to **System.Net.Sockets.Socket.ReceiveFrom**(*buffer*, 0, *buffer*.Length, *socketFlags*, *remoteEP*).

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* or *remoteEP* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* specified an invalid value. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	**Description**
System.Net.SocketPermission	Requires permission to accept connections from the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Accept**.

Socket.ReceiveFrom(System.Byte[], System.Net.EndPoint&) Method

```
[ILASM]
.method public hidebysig instance int32 ReceiveFrom(class
System.Byte[] buffer, class System.Net.EndPoint& remoteEP)

[C#]
public int ReceiveFrom(byte[] buffer, ref EndPoint
remoteEP)
```

Summary

Receives data from a socket and, for connectionless protocols, stores the endpoint associated with the socket that sent the data.

Parameters

Parameter	Description
buffer	A **System.Byte** array to store data received from the socket.
remoteEP	A reference to the **System.Net.EndPoint** associated with the socket that sent the data.

Return Value

A **System.Int32** containing the number of bytes received.

Description

This method is equivalent to **System.Net.Sockets.Socket.ReceiveFrom**(*buffer*, 0, *buffer*.Length, **System.Net.Sockets.SocketFlags.None**, *remoteEP*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* or *remoteEP* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information

	on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	**Description**
System.Net.SocketPermission	Requires permission to accept connections from the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Accept**.

Socket.Select(System.Collections.IList, System.Collections.IList, System.Collections.IList, System.Int32) Method

```
[ILASM]
.method public hidebysig static void Select(class
System.Collections.IList checkRead, class
System.Collections.IList checkWrite, class
System.Collections.IList checkError, int32 microSeconds)

[C#]
public static void Select(IList checkRead, IList
checkWrite, IList checkError, int microSeconds)
```

Summary

Determines the read, write, or error status of a set of **System.Net.Sockets.Socket** instances.

Parameters

Parameter	Description
checkRead	A **System.Collections.IList** object containing the **System.Net.Sockets.Socket** instances to check for read status.
checkWrite	A **System.Collections.IList** object containing the **System.Net.Sockets.Socket** instances to check for write status.
checkError	A **System.Collections.IList** object containing the **System.Net.Sockets.Socket** instances to check for error status.
microSeconds	A **System.Int32** that specifies the time to wait for a response, in microseconds. Specify a negative value to wait indefinitely for the status to be determined.

Description

Upon successful completion, this method removes all **System.Net.Sockets.Socket** instances from the specified list that do not satisfy one of the conditions associated with that list. The following table describes the conditions for each list.

List	Condition to remain in list
checkRead	Data is available for reading (includes out-of-band data if the **System.Net.Sockets.SocketOptionName.OutOfBandInline** value defined in the **System.Net.Sockets.SocketOptionName** enumeration

	is set). -or- The socket is in the listening state with a pending connection, and the **System.Net.Sockets.Socket.Accept** method has been called and is guaranteed to succeed without blocking. -or- The connection has been closed, reset, or terminated.
checkWrite	Data can be sent. -or- A non-blocking **System.Net.Sockets.Socket.Connect** method is being processed and the connection has succeeded.
checkError	The **System.Net.Sockets.SocketOptionName.OutOfBandInline** value defined in the **System.Net.Sockets.SocketOptionName** enumeration is not set and out-of-band data is available. -or- A non-blocking **System.Net.Sockets.Socket.Connect** method is being processed and the connection has failed.

[*Note:* To determine the status of a specific **System.Net.Sockets.Socket** instance, check whether the instance remains in the list after the method returns.]

When the method cannot determine the status of all the **System.Net.Sockets.Socket** instances within the time specified in the *microseconds* parameter, the method removes all the **System.Net.Sockets.Socket** instances from all the lists and returns.

At least one of *checkRead*, *checkWrite*, or *checkError*, is required to contain at least one **System.Net.Sockets.Socket** instance. The other parameters can be empty or **null**.

Exceptions

Exception	Condition
System.ArgumentNullException	All of the following parameters are **null** or empty: *checkRead*, *checkWrite*, and *checkError*.

System.Net.Sockets.SocketException	An error occurred while accessing one of the sockets. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]

Example

The following example determines the status of the socket instance named socket3 and writes the result to the console.

[C#]

```
using System;
using System.Collections;
using System.Net.Sockets;

class SelectTest {

  public static void Main() {

    Socket socket1 =
      new Socket(AddressFamily.InterNetwork,
                 SocketType.Stream,
                 ProtocolType.Tcp);
    Socket socket2 =
      new Socket(AddressFamily.InterNetwork,
                 SocketType.Stream,
                 ProtocolType.Tcp);
    Socket socket3 =
      new Socket(AddressFamily.InterNetwork,
                 SocketType.Stream,
                 ProtocolType.Tcp);

    ArrayList readList = new ArrayList();
    ArrayList writeList = new ArrayList();
    ArrayList errorList = new ArrayList();

    readList.Add(socket1);
    readList.Add(socket2);
    readList.Add(socket3);
    errorList.Add(socket1);
    errorList.Add(socket3);

    // readList.Contains(Socket3) returns true
    // if Socket3 is in ReadList.
    Console.WriteLine(
      "socket3 is placed in readList and errorList.");
    Console.WriteLine(
      "socket3 is {0}in readList.",
      readList.Contains(socket3) ? "": "not ");
    Console.WriteLine(
      "socket3 is {0}in writeList.",
      writeList.Contains(socket3) ? "": "not ");
    Console.WriteLine(
      "socket3 is {0}in errorList.",
      errorList.Contains(socket3) ? "": "not ");

    Socket.Select(readList, writeList, errorList, 10);
    Console.WriteLine("The Select method has been
called.");
    Console.WriteLine(
      "socket3 is {0}in readList.",
      readList.Contains(socket3) ? "": "not ");
    Console.WriteLine(
      "socket3 is {0}in writeList.",
      writeList.Contains(socket3) ? "": "not ");
    Console.WriteLine(
      "socket3 is {0}in errorList.",
      errorList.Contains(socket3) ? "": "not ");
  }
}
```

The output is

```
socket3 is placed in readList and errorList.

socket3 is in readList.

socket3 is not in writeList.

socket3 is in errorList.

The Select method has been called.
```

```
socket3 is not in readList.


socket3 is not in writeList.


socket3 is not in errorList.
```

Socket.Send(System.Byte[], System.Int32, System.Net.Sockets.SocketFlags) Method

[ILASM]

```
.method public hidebysig instance int32 Send(class
System.Byte[] buffer, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags)
```

[C#]

```
public int Send(byte[] buffer, int size, SocketFlags
socketFlags)
```

Summary

Sends data to a connected socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array containing data to send to the socket.
size	A **System.Int32** containing the number of bytes to send.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method is equivalent to **System.Net.Sockets.Socket.Send**(*buffer*, 0, *size*, *socketFlags*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* is **null**.
System.ArgumentOutOfRangeException	*size* < 0.

	-or- *size* > *buffer*.Length.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.Send(System.Byte[], System.Net.Sockets.SocketFlags) Method

[ILASM]

```
.method public hidebysig instance int32 Send(class
System.Byte[] buffer, valuetype
System.Net.Sockets.SocketFlags socketFlags)
```

[C#]

```
public int Send(byte[] buffer, SocketFlags socketFlags)
```

Summary

Sends data to a connected socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array containing data to send to the socket.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method is equivalent to **System.Net.Sockets.Socket.Send**(*buffer*, 0, *buffer*.Length, *socketFlags*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of

	values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.Send(System.Byte[]) Method

```
[ILASM]
.method public hidebysig instance int32 Send(class
System.Byte[] buffer)

[C#]
public int Send(byte[] buffer)
```

Summary

Sends data to a connected socket.

Parameters

Parameter	**Description**
buffer	A **System.Byte** array containing data to send to the socket.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method is equivalent to **System.Net.Sockets.Socket.Send**(*buffer*, 0, *buffer*.Length, **System.Net.Sockets.SocketFlags.None**).

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.Send(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags) Method

```
[ILASM]
.method public hidebysig instance int32 Send(class
System.Byte[] buffer, int32 offset, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags)

[C#]
public int Send(byte[] buffer, int offset, int size,
SocketFlags socketFlags)
```

Summary

Sends data to a connected socket.

Parameters

Parameter	Description
buffer	A **System.Byte** array containing data to send to the socket.
offset	A **System.Int32** that specifies the zero-based position in buffer that is the starting location of the data to send.
size	A **System.Int32** containing the number of bytes to send.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

For connection-oriented protocols, the **System.Net.Sockets.Socket.LocalEndPoint** property of the current instance is required to be set before calling this method.

For connectionless protocols, calling the **System.Net.Sockets.Socket.Connect** methods sets the **System.Net.Sockets.Socket.RemoteEndPoint** property and allows the **System.Net.Sockets.Socket.Send** method to be used instead of the **System.Net.Sockets.Socket.SendTo** method.

When the **System.Net.Sockets.SocketFlags.DontRoute** flag is specified as part of the *socketFlags* parameter, the sent data is not routed.

When the **System.Net.Sockets.SocketFlags.OutOfBand** flag is specified as part of the *socketFlags* parameter, only out-of-band (OOB) data is sent.

When the **System.Net.Sockets.Socket.Blocking** property of the current instance is set to **true** and buffer space is not available within the underlying protocol, this method blocks.

For message-oriented sockets, when the size of *buffer* is greater than the maximum message size of the underlying protocol, no data is sent and the **System.Net.Sockets.SocketException** exception is thrown.

Exceptions

Exception	**Condition**
System.ArgumentNullException	*buffer* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or- *size* < 0. -or- *size* > *buffer*.Length - *offset*.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException**

	class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.SendTo(System.Byte[], System.Int32, System.Int32, System.Net.Sockets.SocketFlags, System.Net.EndPoint) Method

```
[ILASM]
.method public hidebysig instance int32 SendTo(class
System.Byte[] buffer, int32 offset, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags, class
System.Net.EndPoint remoteEP)

[C#]
public int SendTo(byte[] buffer, int offset, int size,
SocketFlags socketFlags, EndPoint remoteEP)
```

Summary

Sends data to the socket associated with the specified endpoint.

Parameters

Parameter	Description
buffer	A **System.Byte** array containing data to send to the socket.
offset	A **System.Int32** that specifies the zero-based position in buffer that is the starting location of the data to send.
size	A **System.Int32** containing the number of bytes to send.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.
remoteEP	The **System.Net.EndPoint** associated with the socket to receive the data.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

For connection-oriented protocols and connected sockets using connectionless protocols, *remoteEP* overrides the endpoint specified in the **System.Net.Sockets.Socket.RemoteEndPoint** property.

For unconnected sockets using connectionless protocols, this method sets the **System.Net.Sockets.Socket.LocalEndPoint** property of the current instance to a value determined by the protocol. Subsequent data is required to be received on **LocalEndPoint**.

When the **System.Net.Sockets.SocketFlags.DontRoute** flag is specified as part of the *socketFlags* parameter, the sent data is not routed.

When the **System.Net.Sockets.SocketFlags.OutOfBand** flag is specified as part of the *socketFlags* parameter, only out-of-band (OOB) data is sent.

When the **System.Net.Sockets.Socket.Blocking** property of the current instance is set to **true** and buffer space is not available within the underlying protocol, this method blocks.

For message-oriented sockets, when the size of *buffer* is greater than the maximum message size of the underlying protocol, no data is sent and the **System.Net.Sockets.SocketException** exception is thrown.

For connection-oriented sockets, the *remoteEP* property is ignored.

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer or remoteEP* is **null**.
System.ArgumentOutOfRangeException	*offset* < 0. -or- *offset* > *buffer*.Length. -or- *size* < 0. -or- *size* > *buffer*.Length - *offset*.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values.

	-or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.SendTo(System.Byte[], System.Int32, System.Net.Sockets.SocketFlags, System.Net.EndPoint) Method

```
[ILASM]
.method public hidebysig instance int32 SendTo(class
System.Byte[] buffer, int32 size, valuetype
System.Net.Sockets.SocketFlags socketFlags, class
System.Net.EndPoint remoteEP)

[C#]
public int SendTo(byte[] buffer, int size, SocketFlags
socketFlags, EndPoint remoteEP)
```

Summary

Sends data to the socket associated with the specified endpoint.

Parameters

Parameter	Description
buffer	A **System.Byte** array containing data to send to the socket.
size	A **System.Int32** containing the number of bytes to send.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.
remoteEP	The **System.Net.EndPoint** associated with the socket to receive the data.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method is equivalent to **System.Net.Sockets.Socket.SendTo**(*buffer*, 0, *size*, *socketFlags*, *remoteEP*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer or remoteEP* is **null**.
System.ArgumentOutOfRangeException	*size* < 0. -or- *size* > *buffer*.Length.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.SendTo(System.Byte[], System.Net.Sockets.SocketFlags, System.Net.EndPoint) Method

[ILASM]

```
.method public hidebysig instance int32 SendTo(class
System.Byte[] buffer, valuetype
System.Net.Sockets.SocketFlags socketFlags, class
System.Net.EndPoint remoteEP)
```

[C#]

```
public int SendTo(byte[] buffer, SocketFlags socketFlags,
EndPoint remoteEP)
```

Summary

Sends data to the socket associated with the specified endpoint.

Parameters

Parameter	Description
buffer	A **System.Byte** array containing data to send to the socket.
socketFlags	A bitwise combination of any of the following values defined in the **System.Net.Sockets.SocketFlags** enumeration: **System.Net.Sockets.SocketFlags.None**, **System.Net.Sockets.SocketFlags.DontRoute**, or **System.Net.Sockets.SocketFlags.OutOfBand**.
remoteEP	The **System.Net.EndPoint** associated with the socket to receive the data.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method is equivalent to **System.Net.Sockets.Socket.SendTo**(*buffer*, 0, *buffer*.Length, *socketFlags*, *remoteEP*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer or remoteEP* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	*socketFlags* is not a valid combination of values. -or- An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.SendTo(System.Byte[], System.Net.EndPoint) Method

[ILASM]

```
.method public hidebysig instance int32 SendTo(class
System.Byte[] buffer, class System.Net.EndPoint remoteEP)
```

[C#]

```
public int SendTo(byte[] buffer, EndPoint remoteEP)
```

Summary

Sends data to the socket associated with the specified endpoint.

Parameters

Parameter	Description
buffer	A **System.Byte** array containing data to send to the socket.
remoteEP	The **System.Net.EndPoint** associated with the socket to receive the data.

Return Value

A **System.Int32** containing the number of bytes sent.

Description

This method is equivalent to **System.Net.Sockets.Socket.SendTo**(*buffer*, 0, *buffer*.Length, **System.Net.Sockets.SocketFlags.None**, *remoteEP*).

Exceptions

Exception	Condition
System.ArgumentNullException	*buffer or remoteEP* is **null**.
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see

	the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by *remoteEP*. See **System.Net.NetworkAccess.Connect**.

Socket.SetSocketOption(System.Net.Sockets.SocketOptionLevel, System.Net.Sockets.SocketOptionName, System.Int32) Method

```
[ILASM]
.method public hidebysig instance void
SetSocketOption(valuetype
System.Net.Sockets.SocketOptionLevel optionLevel, valuetype
System.Net.Sockets.SocketOptionName optionName, int32
optionValue)

[C#]
public void SetSocketOption(SocketOptionLevel optionLevel,
SocketOptionName optionName, int optionValue)
```

Summary

Sets socket options with values of type **System.Int32** and **System.Boolean**.

Parameters

Parameter	Description
optionLevel	One of the values defined in the **System.Net.Sockets.SocketOptionLevel** enumeration.
optionName	One of the values defined in the **System.Net.Sockets.SocketOptionName** enumeration.
optionValue	A **System.Int32** containing the value of the option.

Description

Socket options determine the behavior of the current instance. Multiple options can be set on the current instance by calling this method multiple times.

For a socket option with a **System.Boolean** data type, specify a non-zero *optionValue* to enable the option, and an *optionValue* equal to zero to disable the option.

Socket options are grouped by level of protocol support. The following tables list the members of the **System.Net.Sockets.SocketOptionName** enumeration supported by each member of the **System.Net.Sockets.SocketOptionLevel** enumeration. Only members that have associated values of the

System.Int32 and **System.Boolean** data types are listed.

The following table lists the members of the **System.Net.Sockets.SocketOptionName** enumeration supported by the **Socket** member of the **System.Net.Sockets.SocketOptionLevel** enumeration. Options that do not require permission to access unmanaged code are noted.

SocketOptionName	Description
Broadcast	A **System.Boolean** where **true** indicates broadcast messages are allowed to be sent to the socket.
Debug	A **System.Boolean** where **true** indicates to record debugging information.
DontLinger	A **System.Boolean** where **true** indicates to close the socket without lingering. This option does not require permission to access unmanaged code.
DontRoute	A **System.Boolean** where **true** indicates not to route data.
Error	A **System.Int32** that contains the error code associated with the last socket error. The error code is cleared by this option. This option is read-only.
KeepAlive	A **System.Boolean** where **true** (the default) indicates to enable keep-alives, which allows a connection to remain open after a request has completed. This option does not require permission to access unmanaged code.
OutOfBandInline	A **System.Boolean** where **true** indicates to receive out-of-band data in the normal data stream.
ReceiveBuffer	A **System.Int32** that specifies the total per-socket buffer space reserved for receives. This option does not require permission to access unmanaged code.
ReceiveTimeout	A **System.Int32** that specifies the maximum time, in milliseconds, the **System.Net.Sockets.Socket.Receive** and **System.Net.Sockets.Socket.ReceiveFrom** methods will block when attempting to receive data. If data is not received within this time, a **System.Net.Sockets.SocketException** exception is thrown. This option does not require permission to access unmanaged code.
ReuseAddress	A **System.Boolean** where **true** allows the socket to be bound to an address that is already in use.
SendBuffer	A **System.Int32** that specifies the total per-socket buffer space reserved for sends. This option does not require permission to access unmanaged code.
SendTimeout	A **System.Int32** that specifies the maximum time, in milliseconds, the **System.Net.Sockets.Socket.Send** and **System.Net.Sockets.Socket.SendTo** methods will block when attempting to send data. If data is not sent within this time, a **System.Net.Sockets.SocketException** exception is thrown. This option does not require permission to access unmanaged code.

Type	One of the values defined in the **System.Net.Sockets.SocketType** enumeration. This option is read-only.

The following table lists the members of the **System.Net.Sockets.SocketOptionName** enumeration supported by the **IP** member of the **System.Net.Sockets.SocketOptionLevel** enumeration. These options require permission to access unmanaged code.

SocketOptionName	**Description**
HeaderIncluded	A **System.Boolean** where **true** indicates the application is providing the IP header for outgoing datagrams.
IPOptions	A **System.Byte** array that specifies IP options to be inserted into outgoing datagrams.
IpTimeToLive	A **System.Int32** that specifies the time-to-live for datagrams. The time-to-live designates the number of networks on which the datagram is allowed to travel before being discarded by a router.
MulticastInterface	A **System.Byte** array that specifies the interface for outgoing multicast packets.
MulticastLoopback	A **System.Boolean** where **true** enables multicast loopback.
MulticastTimeToLive	A **System.Int32** that specifies the time-to-live for multicast datagrams.
TypeOfService	A **System.Int32** that specifies the type of service field in the IP header.
UseLoopback	A **System.Boolean** where **true** indicates to send a copy of the data back to the sender.

The following table lists the members of the **System.Net.Sockets.SocketOptionName** enumeration supported by the **Tcp** member of the **System.Net.Sockets.SocketOptionLevel** enumeration. These options do not require permission to access unmanaged code.

SocketOptionName	**Description**
BsdUrgent	A **System.Boolean** where **true** indicates to use urgent data as defined by IETF RFC 1222. Once enabled, this option cannot be disabled.
Expedited	A **System.Boolean** where **true** indicates to use expedited data as defined by IETF RFC 1222. Once enabled, this option cannot be disabled.

NoDelay	A **System.Boolean** where **true** indicates to disable the Nagle algorithm for send coalescing.

The following table lists the members of the **System.Net.Sockets.SocketOptionName** enumeration supported by the **Udp** member of the **System.Net.Sockets.SocketOptionLevel** enumeration. These options do not require permission to access unmanaged code.

SocketOptionName	**Description**
ChecksumCoverage	A **System.Boolean** that specifies UDP checksum coverage.
NoChecksum	A **System.Boolean** where **true** indicates to send UDP datagrams with the checksum set to zero.

[*Note:* For the **AddMembership**, **DropMembership**, and **Linger** members of the **System.Net.Sockets.SocketOptionName** enumeration, see the **System.Net.Sockets.Socket.SetSocketOption**(**System.Net.Sockets.SocketOptionLevel**, **System.Net.Sockets.SocketOptionName**, **System.Object**) version of this method.]

Exceptions

Exception	**Condition**
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	**Description**
System.Security.Permissions.	Some options require permission to access unmanaged

SecurityPermission	code. All the options that do not require permission are noted in the tables in the Description section. All options not so noted require this permission. See **System.Security.Permissions.SecurityPermissionFlag UnmanagedCode**.

Socket.SetSocketOption(System.Net.Sockets.SocketOptionLevel, System.Net.Sockets.SocketOptionName, System.Byte[]) Method

```
[ILASM]
.method public hidebysig instance void
SetSocketOption(valuetype
System.Net.Sockets.SocketOptionLevel optionLevel, valuetype
System.Net.Sockets.SocketOptionName optionName, class
System.Byte[] optionValue)

[C#]
public void SetSocketOption(SocketOptionLevel optionLevel,
SocketOptionName optionName, byte[] optionValue)
```

Summary

Sets socket options with values of type **Byte[]**.

Parameters

Parameter	Description
optionLevel	One of the values defined in the **System.Net.Sockets.SocketOptionLevel** enumeration.
optionName	One of the values defined in the **System.Net.Sockets.SocketOptionName** enumeration.
optionValue	A **System.Byte** array containing the value of the option.

Description

Socket options determine the behavior of the current instance. Multiple options can be set on the current instance by calling this method multiple times.

[*Note:* For socket options with values of type **System.Int32** or **System.Boolean**, see the **System.Net.Sockets.Socket.SetSocketOption**(**System.Net.Sockets.SocketOptionLevel**, **System.Net.Sockets.SocketOptionName**, **System.Int32**) version of this method.]

[*Note:* For the **System.Net.Sockets.SocketOptionName.AddMembership**, **System.Net.Sockets.SocketOptionName.DropMembership**, or **System.Net.Sockets.SocketOptionName.Linger** socket options, see the

System.Net.Sockets.Socket.SetSocketOption(**System.Net.Sockets.SocketOptionLevel**, **System.Net.Sockets.SocketOptionName**, **System.Object**) version of this method.]

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	Requires permission to access unmanaged code. See **System.Security.Permissions.SecurityPermissionFlag** **UnmanagedCode**.

Socket.SetSocketOption(System.Net.Sockets.SocketOptionLevel, System.Net.Sockets.SocketOptionName, System.Object) Method

```
[ILASM]
.method public hidebysig instance void
SetSocketOption(valuetype
System.Net.Sockets.SocketOptionLevel optionLevel, valuetype
System.Net.Sockets.SocketOptionName optionName, object
optionValue)

[C#]
public void SetSocketOption(SocketOptionLevel optionLevel,
SocketOptionName optionName, object optionValue)
```

Summary

Sets the **System.Net.Sockets.SocketOptionName.AddMembership**, **System.Net.Sockets.SocketOptionName.DropMembership**, or **System.Net.Sockets.SocketOptionName.Linger** socket options.

Parameters

Parameter	Description
optionLevel	Either the **Socket** or **IP** member of the **System.Net.Sockets.SocketOptionLevel** enumeration.
optionName	Either the **Linger**, **AddMembership**, or **DropMembership** member of the **System.Net.Sockets.SocketOptionName** enumeration.
optionValue	An instance of the **System.Net.Sockets.LingerOption** or **System.Net.Sockets.MulticastOption** class.

Description

Socket options determine the behavior of the current instance. Multiple options can be set on the current instance by calling this method multiple times.

The following table summarizes the valid combinations of input parameters.

optionLevel/optionName	optionValue
Socket/Linger	An instance of the **System.Net.Sockets.LingerOption** class.

IP/AddMembership - or - **IP/DropMembership**	An instance of the **System.Net.Sockets.MulticastOption** class.

When setting the **System.Net.Sockets.SocketOptionName.Linger** option, a **System.ArgumentException** exception is thrown if the **System.Net.Sockets.LingerOption.LingerTime** property of the **System.Net.Sockets.LingerOption** instance is less than zero or greater than **System.UInt16.MaxValue**.

[*Note:* For more information on the **System.Net.Sockets.SocketOptionName.Linger** option, see the **System.Net.Sockets.LingerOption** class and the **System.Net.Sockets.Socket.Shutdown** method.

For more information on the **System.Net.Sockets.SocketOptionName.AddMembership** and **System.Net.Sockets.SocketOptionName.DropMembership** options, see the **System.Net.Sockets.MulticastOption** class.

For socket options with values of type **System.Int32** or **System.Boolean**, see the **System.Net.Sockets.Socket.SetSocketOption**(**System.Net.Sockets.SocketOptionLevel**, **System.Net.Sockets.SocketOptionName**, **System.Int32**) version of this method.]

Exceptions

Exception	Condition
System.ArgumentException	*optionLevel*, *optionName*, or *optionValue* specified an invalid value.
System.ArgumentNullException	*optionValue* is **null**.
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.Security.SecurityException	A caller in the call stack does not have the required permissions.
System.ObjectDisposedException	The current instance has been disposed.

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	The **System.Net.Sockets.SocketOptionName.AddMember** and **System.Net.Sockets.SocketOptionName.DropMembe** options require permission to access unmanaged code. S **System.Security.Permissions.SecurityPermissionFla** **UnmanagedCode**.

Socket.Shutdown(System.Net.Sockets.SocketShutdown) Method

```
[ILASM]
.method public hidebysig instance void Shutdown(valuetype
System.Net.Sockets.SocketShutdown how)

[C#]
public void Shutdown(SocketShutdown how)
```

Summary

Terminates the ability to send or receive data on a connected socket.

Parameters

Parameter	Description
how	One of the values defined in the **System.Net.Sockets.SocketShutdown** enumeration.

Description

When *how* is set to **System.Net.Sockets.SocketShutdown.Send**, the socket on the other end of the connection is notified that the current instance will not send any more data. If the **System.Net.Sockets.Socket.Send** method is subsequently called, a **System.Net.Sockets.SocketException** exception is thrown.

When *how* is set to **System.Net.Sockets.SocketShutdown.Receive**, the socket on the other end of the connection is notified that the current instance will not receive any more data. After all the data currently queued on the current instance is received, any subsequent calls to the **System.Net.Sockets.Socket.Receive** method cause a **System.Net.Sockets.SocketException** exception to be thrown.

Setting *how* to **System.Net.Sockets.SocketShutdown.Both** terminates both sends and receives as described above. Once this occurs, the socket cannot be used.

[*Note:* To free resources allocated by the current instance, call the **System.Net.Sockets.Socket.Close** method.

Expected common usage is for the **System.Net.Sockets.Socket.Shutdown** method to be called before the **System.Net.Sockets.Socket.Close** method to ensure that all pending data is sent or received.]

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.System.IDisposable.Dispose() Method

```
[ILASM]
.method private final hidebysig virtual void
System.IDisposable.Dispose()

[C#]
void IDisposable.Dispose()
```

Summary

Implemented to support the **System.IDisposable** interface. [Note: For more information, see **System.IDisposable.Dispose**.]

Socket.AddressFamily Property

```
[ILASM]
.property valuetype System.Net.Sockets.AddressFamily
AddressFamily { public hidebysig specialname instance
valuetype System.Net.Sockets.AddressFamily
get_AddressFamily() }

[C#]
public AddressFamily AddressFamily { get; }
```

Summary

Gets the address family of the current instance.

Property Value

One of the values defined in the **System.Net.Sockets.AddressFamily** enumeration.

Description

This property is read-only.

This property is set by the constructor for the current instance. The value of this property specifies the addressing scheme used by the current instance to resolve an address.

Socket.Available Property

[ILASM]
```
.property int32 Available { public hidebysig specialname
instance int32 get_Available() }
```

[C#]
```
public int Available { get; }
```

Summary

Gets the amount of data available to be read in a single **System.Net.Sockets.Socket.Receive** or **System.Net.Sockets.Socket.ReceiveFrom** call.

Property Value

A **System.Int32** containing the number of bytes of data that are available to be read.

Description

This property is read-only.

When the current instance is stream-oriented (for example, the **System.Net.Sockets.SocketType.Stream** socket type), the available data is generally the total amount of data queued on the current instance.

When the current instance is message-oriented (for example, the **System.Net.Sockets.SocketType.Dgram** socket type), the available data is the first message in the input queue.

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.Blocking Property

[ILASM]
```
.property bool Blocking { public hidebysig specialname
instance bool get_Blocking() public hidebysig specialname
instance void set_Blocking(bool value) }
```

[C#]
```
public bool Blocking { get; set; }
```

Summary

Gets or sets a **System.Boolean** value that indicates whether the socket is in blocking mode.

Property Value

true indicates that the current instance is in blocking mode; **false** indicates that the current instance is in non-blocking mode.

Description

Blocking is when a method waits to complete an operation before returning. Sockets are created in blocking mode by default.

Except for when the current instance has been disposed, no notification is given when an attempt to change the value of this property fails.

Exceptions

Exception	Condition
System.ObjectDisposedException	The current instance has been disposed.

Socket.Connected Property

[ILASM]

```
.property bool Connected { public hidebysig specialname
instance bool get_Connected() }
```

[C#]

```
public bool Connected { get; }
```

Summary

Gets a **System.Boolean** value indicating whether the current instance is connected.

Property Value

true indicates that the current instance was connected at the time of the last I/O operation; **false** indicates that the current instance is not connected.

Description

This property is read-only.

When this property returns **true**, the current instance was connected at the time of the last I/O operation; it might not still be connected. When this property returns **false**, the current instance was never connected or is not currently connected.

The current instance is considered connected when the **System.Net.Sockets.Socket.RemoteEndPoint** property contains a valid endpoint.

[*Note:* The **System.Net.Sockets.Socket.Accept** and **System.Net.Sockets.Socket.Connect** methods, and their asynchronous counterparts set this property.]

The following member must be implemented if the RuntimeInfrastructure library is present in the implementation.

Socket.Handle Property

[ILASM]

```
.property valuetype System.IntPtr Handle { public hidebysig
specialname instance valuetype System.IntPtr get_Handle() }
```

[C#]

```
public IntPtr Handle { get; }
```

Summary

Gets the operating system handle for the current instance.

Property Value

A **System.IntPtr** containing the operating system handle for the current instance.

Description

This property is read-only.

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	Requires permission to access unmanaged code. See **System.Security.Permissions.SecurityPermissionFlag** **UnmanagedCode**.

Socket.LocalEndPoint Property

[ILASM]

```
.property class System.Net.EndPoint LocalEndPoint { public
hidebysig specialname instance class System.Net.EndPoint
get_LocalEndPoint() }
```

[C#]

```
public EndPoint LocalEndPoint { get; }
```

Summary

Gets the local endpoint associated with the current instance.

Property Value

The local **System.Net.EndPoint** associated with the current instance.

Description

This property is read-only.

This property contains the network connection information for the current instance.

[*Note:* The **System.Net.Sockets.Socket.Bind** and **System.Net.Sockets.Socket.Accept** methods, and their asynchronous counterparts set this property. If not previously set, the **System.Net.Sockets.Socket.Connect** and **System.Net.Sockets.Socket.SendTo** methods, and their asynchronous counterparts set this property.]

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.ProtocolType Property

[ILASM]

```
.property valuetype System.Net.Sockets.ProtocolType
ProtocolType { public hidebysig specialname instance
valuetype System.Net.Sockets.ProtocolType
get_ProtocolType() }
```

[C#]

```
public ProtocolType ProtocolType { get; }
```

Summary

Gets the protocol type of the current instance.

Property Value

One of the values defined in the **System.Net.Sockets.ProtocolType** enumeration.

Description

This property is read-only.

This property is set by the constructor for the current instance. The value of this property specifies the protocol used by the current instance.

Socket.RemoteEndPoint Property

[ILASM]

```
.property class System.Net.EndPoint RemoteEndPoint { public
hidebysig specialname instance class System.Net.EndPoint
get_RemoteEndPoint() }
```

[C#]

```
public EndPoint RemoteEndPoint { get; }
```

Summary

Gets the remote endpoint associated with the current instance.

Property Value

The remote **System.Net.EndPoint** associated with the current instance.

Description

This property is read-only.

This property contains the network connection information associated with the socket communicating with the current instance.

There is no remote endpoint associated with a socket in the listening state. An attempt to access the **System.Net.Sockets.Socket.RemoteEndPoint** property causes a **System.Net.Sockets.SocketException** exception to be thrown.

[*Note:* The **System.Net.Sockets.Socket.Accept** and **System.Net.Sockets.Socket.Connect** methods, and their asynchronous counterparts set this property.]

Exceptions

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [*Note:* For additional information on causes of the **SocketException**, see the **System.Net.Sockets.SocketException** class.]
System.ObjectDisposedException	The current instance has been disposed.

Socket.SocketType Property

```
[ILASM]
.property valuetype System.Net.Sockets.SocketType
SocketType { public hidebysig specialname instance
valuetype System.Net.Sockets.SocketType get_SocketType() }

[C#]
public SocketType SocketType { get; }
```

Summary

Gets the socket type of the current instance.

Property Value

One of the values defined in the **System.Net.Sockets.SocketType** enumeration.

Description

This property is read-only.

This property is set by the constructor for the current instance.