

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]
- *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

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

3
4 A **System.Net.Sockets.Socket** instance has a local and a remote
5 endpoint associated with it. The local endpoint contains the connection
6 information for the current socket instance. The remote endpoint
7 contains the connection information for the socket that the current
8 instance communicates with. The endpoints are required to be an
9 instance of a type derived from the **System.Net.EndPoint** class. For
10 the Transmission Control Protocol (TCP) and User Datagram Protocol
11 (UDP) protocols, an endpoint includes the address family, an Internet
12 Protocol (IP) address, and a port number. For connection-oriented
13 protocols (for example, TCP), the remote endpoint does not have to be
14 specified when transferring data. For connectionless protocols (for
15 example, UDP), the remote endpoint is required to be specified.

16
17 Methods are provided for both synchronous and asynchronous
18 operations. A synchronous method can operate in blocking mode, in
19 which it waits (blocks) until the operation is complete before returning,
20 or in non-blocking mode, where it returns immediately, possibly before
21 the operation has completed. The blocking mode is set through the
22 **System.Net.Sockets.Socket.Blocking** property.

23
24 An asynchronous method returns immediately and, by convention,
25 relies on a delegate to complete the operation. Asynchronous methods
26 have names which correspond to their synchronous counterparts
27 prefixed with either 'Begin' or 'End'. For example, the synchronous
28 **System.Net.Sockets.Socket.Accept** method has asynchronous
29 counterpart methods named
30 **System.Net.Sockets.Socket.BeginAccept** and
31 **System.Net.Sockets.Socket.EndAccept**. The example for the
32 **System.Net.Sockets.Socket.BeginAccept** method shows the basic
33 steps for using an asynchronous operation. A complete working
34 example follows this discussion.

35
36 Connection-oriented protocols commonly use the client/server model.
37 In this model, one of the sockets is set up as a server, and one or
38 more sockets are set up as clients. A general procedure demonstrating
39 the synchronous communication process for this model is as follows.

40
41 On the server-side:

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

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

10 On the client-side:

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

22 The shutdown step in the previous procedure is not necessary but

23 ensures that any pending data is not lost. If the

24 **System.Net.Sockets.Socket.Shutdown** method is not called, the

25 **System.Net.Sockets.Socket.Close** method shuts down the

26 connection either gracefully or by force. A graceful closure attempts to

27 transfer all pending data before the connection is terminated. Use the

28 **System.Net.Sockets.SocketOptionName.Linger** socket option to

29 specify a graceful closure for a socket.

30

31 [*Note:* This implementation is based on the UNIX sockets

32 implementation in the Berkeley Software Distribution (BSD, release

33 4.3) from the University of California at Berkeley.]

34 **Example**

35

36 The following examples provide a client/server application that

37 demonstrates the use of asynchronous communication between

38 sockets. Run the client and server on different consoles.

39

40 The following code is for the server application. Start this application

41 before the client application.

42

43

[C#]

44

```
using System;
```

45

```
using System.Threading;
```

46

```
using System.Text;
```

```

1      using System.Net;
2      using System.Net.Sockets;
3
4      public class Server
5      {
6          // used to pass state information to delegate
7          internal class StateObject
8          {
9              internal byte[] sBuffer;
10             internal Socket sSocket;
11             internal StateObject(int size, Socket sock) {
12                 sBuffer = new byte[size];
13                 sSocket = sock;
14             }
15         }
16         static void Main()
17         {
18             IPAddress ipAddress =
19                 Dns.Resolve(Dns.GetHostName()).AddressList[0];
20
21             IPEndPoint ipEndpoint =
22                 new IPEndPoint(ipAddress, 1800);
23
24             Socket listenSocket =
25                 new Socket(AddressFamily.InterNetwork,
26                             SocketType.Stream,
27                             ProtocolType.Tcp);
28
29             listenSocket.Bind(ipEndpoint);
30             listenSocket.Listen(1);
31             IAsyncResult asyncAccept = listenSocket.BeginAccept(
32                 new AsyncCallback(Server.acceptCallback),
33                 listenSocket);
34
35             // could call listenSocket.EndAccept(asyncAccept) here
36             // instead of in the callback method, but since
37             // EndAccept blocks, the behavior would be similar to
38             // calling the synchronous Accept method
39
40             Console.WriteLine("Connection in progress.");
41             if(writeDot(asyncAccept) == true)
42             {
43                 // allow time for callbacks to
44                 // finish before the program ends
45                 Thread.Sleep(3000);
46             }
47         }
48
49         public static void
50         acceptCallback(IAsyncResult asyncAccept) {
51             Socket listenSocket = (Socket)asyncAccept.AsyncState;
52             Socket serverSocket =
53                 listenSocket.EndAccept(asyncAccept);
54
55             // arriving here means the operation completed
56             // (asyncAccept.IsCompleted = true) but not
57             // necessarily successfully

```

```

1         if(serverSocket.Connected == false)
2         {
3             Console.WriteLine(".server is not connected.");
4             return;
5         }
6         else Console.WriteLine(".server is connected.");
7
8         listenSocket.Close();
9
10        StateObject stateObject =
11            new StateObject(16, serverSocket);
12
13        // this call passes the StateObject because it
14        // needs to pass the buffer as well as the socket
15        IAsyncResult asyncReceive =
16            serverSocket.BeginReceive(
17                stateObject.sBuffer,
18                0,
19                stateObject.sBuffer.Length,
20                SocketFlags.None,
21                new AsyncCallback(receiveCallback),
22                stateObject);
23
24        Console.Write("Receiving data.");
25        writeDot(asyncReceive);
26    }
27
28    public static void
29    receiveCallback(IAsyncResult asyncReceive) {
30        StateObject stateObject =
31            (StateObject)asyncReceive.AsyncState;
32        int bytesReceived =
33            stateObject.sSocket.EndReceive(asyncReceive);
34
35        Console.WriteLine(
36            ".{0} bytes received: {1}",
37            bytesReceived.ToString(),
38            Encoding.ASCII.GetString(stateObject.sBuffer));
39
40        byte[] sendBuffer =
41            Encoding.ASCII.GetBytes("Goodbye");
42        IAsyncResult asyncSend =
43            stateObject.sSocket.BeginSend(
44                sendBuffer,
45                0,
46                sendBuffer.Length,
47                SocketFlags.None,
48                new AsyncCallback(sendCallback),
49                stateObject.sSocket);
50
51        Console.Write("Sending response.");
52        writeDot(asyncSend);
53    }
54
55    public static void sendCallback(IAsyncResult asyncSend) {
56        Socket serverSocket = (Socket)asyncSend.AsyncState;
57        int bytesSent = serverSocket.EndSend(asyncSend);

```

```

1         Console.WriteLine(
2             ".{0} bytes sent.{1}{1}Shutting down.",
3             bytesSent.ToString(),
4             Environment.NewLine);
5
6         serverSocket.Shutdown(SocketShutdown.Both);
7         serverSocket.Close();
8     }
9
10    // times out after 20 seconds but operation continues
11    internal static bool writeDot(IAsyncResult ar)
12    {
13        int i = 0;
14        while(ar.IsCompleted == false)
15        {
16            if(i++ > 40)
17            {
18                Console.WriteLine("Timed out.");
19                return false;
20            }
21            Console.Write(".");
22            Thread.Sleep(500);
23        }
24        return true;
25    }
26 }
27

```

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

32
33

[C#]

```

34 using System;
35 using System.Threading;
36 using System.Text;
37 using System.Net;
38 using System.Net.Sockets;
39
40 public class Client {
41
42     // used to pass state information to delegate
43     class StateObject
44     {
45         internal byte[] sBuffer;
46         internal Socket sSocket;
47         internal StateObject(int size, Socket sock) {
48             sBuffer = new byte[size];

```

```

1         sSocket = sock;
2     }
3 }
4
5 static void Main(string[] argHostName)
6 {
7     IPAddress ipAddress =
8         Dns.Resolve(argHostName[0]).AddressList[0];
9
10    IPEndPoint ipEndpoint =
11        new IPEndPoint(ipAddress, 1800);
12
13    Socket clientSocket = new Socket(
14        AddressFamily.InterNetwork,
15        SocketType.Stream,
16        ProtocolType.Tcp);
17
18    IAsyncResult asyncConnect = clientSocket.BeginConnect(
19        ipEndpoint,
20        new AsyncCallback(connectCallback),
21        clientSocket);
22
23    Console.Write("Connection in progress.");
24    if(writeDot(asyncConnect) == true)
25    {
26        // allow time for callbacks to
27        // finish before the program ends
28        Thread.Sleep(3000);
29    }
30 }
31
32 public static void
33 connectCallback(IAsyncResult asyncConnect) {
34     Socket clientSocket =
35         (Socket)asyncConnect.AsyncState;
36     clientSocket.EndConnect(asyncConnect);
37     // arriving here means the operation completed
38     // (asyncConnect.IsCompleted = true) but not
39     // necessarily successfully
40     if(clientSocket.Connected == false)
41     {
42         Console.WriteLine(".client is not connected.");
43         return;
44     }
45     else Console.WriteLine(".client is connected.");
46
47     byte[] sendBuffer = Encoding.ASCII.GetBytes("Hello");
48     IAsyncResult asyncSend = clientSocket.BeginSend(
49         sendBuffer,
50         0,
51         sendBuffer.Length,
52         SocketFlags.None,
53         new AsyncCallback(sendCallback),
54         clientSocket);
55
56     Console.Write("Sending data.");
57     writeDot(asyncSend);

```

```

1      }
2
3      public static void sendCallback(IAsyncResult asyncSend)
4      {
5          Socket clientSocket = (Socket)asyncSend.AsyncState;
6          int bytesSent = clientSocket.EndSend(asyncSend);
7          Console.WriteLine(
8              ".{0} bytes sent.",
9              bytesSent.ToString());
10
11         StateObject stateObject =
12             new StateObject(16, clientSocket);
13
14         // this call passes the StateObject because it
15         // needs to pass the buffer as well as the socket
16         IAsyncResult asyncReceive =
17             clientSocket.BeginReceive(
18                 stateObject.sBuffer,
19                 0,
20                 stateObject.sBuffer.Length,
21                 SocketFlags.None,
22                 new AsyncCallback(receiveCallback),
23                 stateObject);
24
25         Console.Write("Receiving response.");
26         writeDot(asyncReceive);
27     }
28
29     public static void
30     receiveCallback(IAsyncResult asyncReceive) {
31         StateObject stateObject =
32             (StateObject)asyncReceive.AsyncState;
33
34         int bytesReceived =
35             stateObject.sSocket.EndReceive(asyncReceive);
36
37         Console.WriteLine(
38             ".{0} bytes received: {1}{2}{2}Shutting down.",
39             bytesReceived.ToString(),
40             Encoding.ASCII.GetString(stateObject.sBuffer),
41             Environment.NewLine);
42
43         stateObject.sSocket.Shutdown(SocketShutdown.Both);
44         stateObject.sSocket.Close();
45     }
46
47     // times out after 2 seconds but operation continues
48     internal static bool writeDot(IAsyncResult ar)
49     {
50         int i = 0;
51         while(ar.IsCompleted == false)
52         {
53             if(i++ > 20)
54             {
55                 Console.WriteLine("Timed out.");
56                 return false;
57             }

```

```
1         Console.WriteLine(".");
2         Thread.Sleep(100);
3     }
4     return true;
5 }
6 }
7 }
```

8 The output of the server application is

```
9
10 Connection in progress.....server is connected.
11
12
13 Receiving data.....5 bytes received: Hello
14
15
16 Sending response....7 bytes sent.
17
18
19 Shutting down.
```

20 -----

21

22

23

24

25 The output of the client application is

```
26
27 Connection in progress.....client is connected.
28
29
```

```
1      Sending data.....5 bytes sent.  
2  
3  
4      Receiving response.....7 bytes received: Goodbye  
5  
6  
7      Shutting down.  
8  
9
```

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
<i>addressFamily</i>	One of the values defined in the System.Net.Sockets.AddressFamily enumeration.
<i>socketType</i>	One of the values defined in the System.Net.Sockets.SocketType enumeration.
<i>protocolType</i>	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.

1 **Exceptions**
2
3

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

4
5
6

1 Socket.Accept() Method

```
2 [ILASM]  
3 .method public hidebysig instance class  
4 System.Net.Sockets.Socket Accept()  
  
5 [C#]  
6 public Socket Accept()
```

7 Summary

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

10 Return Value

11

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

13 Description

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

19

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

24

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

28

- **System.Net.Sockets.Socket.AddressFamily**

29

- **System.Net.Sockets.Socket.Blocking**

30

- **System.Net.Sockets.Socket.LocalEndPoint**

31

- **System.Net.Sockets.Socket.ProtocolType**

32

- **System.Net.Sockets.Socket.SocketType**

33

34

35

36

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**.

1 **Exceptions**
2
3

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

4
5
6

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

```
3 [ILASM]  
4 .method public hidebysig instance class System.IAsyncResult  
5 BeginAccept(class System.AsyncCallback callback, object  
6 state)  
  
7 [C#]  
8 public IAsyncResult BeginAccept(AsyncCallback callback,  
9 object state)
```

10 Summary

11 Begins an asynchronous operation to accept an incoming connection
12 request.

13 Parameters

Parameter	Description
<i>callback</i>	A System.AsyncCallback delegate, or null .
<i>state</i>	An application-defined object, or null .

16 Return Value

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

21 Description

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

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

31 If the *callback* parameter is not **null**, the method referenced by
32 *callback* is invoked when the asynchronous operation completes. The
33 **System.IAsyncResult** object returned by this method is passed as
34 the argument to the method referenced by *callback*. The method
35 referenced by *callback* can retrieve the results of the operation by
36 calling the **System.Net.Sockets.Socket.EndAccept** method.
37

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

5
6 To determine the connection status, check the
7 **System.Net.Sockets.Socket.Connected** property, or use either the
8 **System.Net.Sockets.Socket.Poll** or
9 **System.Net.Sockets.Socket.Select** method.

10
11 [Note: For more information, see
12 **System.Net.Sockets.Socket.Accept**, the synchronous version of this
13 method.]

14 Exceptions

15
16

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.

17
18
19

Example

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

22
23

[C#]

```
24 public class Server
25 {
26     static void Main()
27     {
28         .
29         .
30         .
31         listenSocket.BeginAccept(
32             new AsyncCallback(Server.acceptCallback),
33             listenSocket);
34         .
35         .
36         .
37         // EndAccept can be called here
38         .
39         .
40     }
```

```
1     }
2
3     public static void
4         acceptCallback(IAsyncResult asyncAccept)
5     {
6         Socket listenSocket =
7             (Socket)asyncAccept.AsyncState;
8
9         Socket serverSocket =
10            listenSocket.EndAccept(asyncAccept);
11
12            serverSocket.BeginReceive(...);
13            .
14            .
15            .
16        }
17    }
18
```

1 Socket.BeginConnect(System.Net.EndPoint, 2 System.AsyncCallback, System.Object) 3 Method

```
4 [ILASM]  
5 .method public hidebysig instance class System.IAsyncResult  
6 BeginConnect(class System.Net.EndPoint remoteEP, class  
7 System.AsyncCallback callback, object state)  
  
8 [C#]  
9 public IAsyncResult BeginConnect(EndPoint remoteEP,  
10 AsyncCallback callback, object state)
```

11 Summary

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

14 Parameters

15
16

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

17
18
19

18 Return Value

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

22 Description

23 To release resources allocated by the
24 **System.Net.Sockets.Socket.BeginConnect** method, call the
25 **System.Net.Sockets.Socket.EndConnect** method, and specify the
26 **System.IAsyncResult** object returned by this method.
27

28 [Note: The **System.Net.Sockets.Socket.EndConnect** method should
29 be called exactly once for each call to the
30 **System.Net.Sockets.Socket.BeginConnect** method.]
31

32 If the *callback* parameter is not **null**, the method referenced by
33 *callback* is invoked when the asynchronous operation completes. The
34 **System.IAsyncResult** object returned by this method is passed as

1 the argument to the method referenced by *callback*. The method
2 referenced by *callback* can retrieve the results of the operation by
3 calling the **System.Net.Sockets.Socket.EndConnect** method.

4
5 The *state* parameter can be any object that the caller wishes to have
6 available for the duration of the asynchronous operation. This object is
7 available via the **System.IAsyncResult.AsyncState** property of the
8 object returned by this method.

9
10 To determine the connection status, check the
11 **System.Net.Sockets.Socket.Connected** property, or use either the
12 **System.Net.Sockets.Socket.Poll** or
13 **System.Net.Sockets.Socket.Select** method.

14
15 [Note: For more information, see
16 **System.Net.Sockets.Socket.Connect**, the synchronous version of
17 this method.]

18 Exceptions

Exception	Condition
System.ArgumentNullException	<i>remoteEP</i> 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.

21 Example

22
23
24 For an outline of an asynchronous operation, see the
25 **System.Net.Sockets.Socket.BeginAccept** method. For the
26 complete example, which uses the
27 **System.Net.Sockets.Socket.BeginConnect** method, see the
28 **System.Net.Sockets.Socket** class overview.

29 Permissions

Permission	Description
------------	-------------

1
2
3

System.Net.SocketPermission	Requires permission to make a connection to the endpoint defined by <i>remoteEP</i> . [Note: See System.Net.NetworkAccess.Connect.]
------------------------------------	---

1 **Socket.BeginReceive(System.Byte[],**
2 **System.Int32, System.Int32,**
3 **System.Net.Sockets.SocketFlags,**
4 **System.AsyncCallback, System.Object)**
5 **Method**

```
6 [ILASM]  
7 .method public hidebysig instance class System.IAsyncResult  
8 BeginReceive(class System.Byte[] buffer, int32 offset,  
9 int32 size, valuetype System.Net.Sockets.SocketFlags  
10 socketFlags, class System.AsyncCallback callback, object  
11 state)  
  
12 [C#]  
13 public IAsyncResult BeginReceive(byte[] buffer, int offset,  
14 int size, SocketFlags socketFlags, AsyncCallback callback,  
15 object state)
```

16 **Summary**

17 Begins an asynchronous operation to receive data from a socket.

18 **Parameters**

19
20

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>offset</i>	A System.Int32 containing the zero-based position in <i>buffer</i> to begin storing the received data.
<i>size</i>	A System.Int32 containing the number of bytes to receive.
<i>socketFlags</i>	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 .
<i>callback</i>	A System.AsyncCallback delegate, or null .
<i>state</i>	An application-defined object, or null .

21
22
23

Return Value

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

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

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

1

2 **Example**

3

4

5

6

7

8

9

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.

1 **Socket.BeginReceiveFrom(System.Byte[],**
2 **System.Int32, System.Int32,**
3 **System.Net.Sockets.SocketFlags,**
4 **System.Net.EndPoint&**
5 **System.AsyncCallback, System.Object)**
6 **Method**

```
7 [ILASM]  
8 .method public hidebysig instance class System.IAsyncResult  
9 BeginReceiveFrom(class System.Byte[] buffer, int32 offset,  
10 int32 size, valuetype System.Net.Sockets.SocketFlags  
11 socketFlags, class System.Net.EndPoint& remoteEP, class  
12 System.AsyncCallback callback, object state)  
  
13 [C#]  
14 public IAsyncResult BeginReceiveFrom(byte[] buffer, int  
15 offset, int size, SocketFlags socketFlags, ref EndPoint  
16 remoteEP, AsyncCallback callback, object state)
```

17 **Summary**

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

21 **Parameters**

22
23

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>offset</i>	A System.Int32 containing the zero-based position in <i>buffer</i> to begin storing the received data.
<i>size</i>	A System.Int32 containing the number of bytes to receive.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	An instance of a class derived from the System.Net.EndPoint class, which contains the endpoint associated with the socket that sent the data.
<i>callback</i>	A System.AsyncCallback delegate, or null .
<i>state</i>	An application-defined object, or null .

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

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	<i>buffer</i> is null . -or- <i>remoteEP</i> is null .
System.ArgumentOutOfRangeException	<i>offset</i> < 0. -or- <i>offset</i> > <i>buffer.Length</i> . -or-

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

1
2
3

Example

4
5
6

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

7
8
9

Permissions

Permission	Description
System.Net.SocketPermission	<p>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.</p> <p>Requires permission to make a connection to the endpoint defined by <i>remoteEP</i>. See System.Net.NetworkAccess.Connect.</p>

10
11
12

1 **Socket.BeginSend(System.Byte[],**
2 **System.Int32, System.Int32,**
3 **System.Net.Sockets.SocketFlags,**
4 **System.AsyncCallback, System.Object)**
5 **Method**

```
6 [ILASM]  
7 .method public hidebysig instance class System.IAsyncResult  
8 BeginSend(class System.Byte[] buffer, int32 offset, int32  
9 size, valuetype System.Net.Sockets.SocketFlags socketFlags,  
10 class System.AsyncCallback callback, object state)  
  
11 [C#]  
12 public IAsyncResult BeginSend(byte[] buffer, int offset,  
13 int size, SocketFlags socketFlags, AsyncCallback callback,  
14 object state)
```

15 **Summary**

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

17 **Parameters**

Parameter	Description
<i>buffer</i>	A System.Byte array storing data to send to the socket.
<i>offset</i>	A System.Int32 containing the zero-based position in <i>buffer</i> containing the starting location of the data to send.
<i>size</i>	A System.Int32 containing the number of bytes to send.
<i>socketFlags</i>	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 .
<i>callback</i>	A System.AsyncCallback delegate, or null .
<i>state</i>	An application-defined object, or null .

20 **Return Value**

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

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

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

1

2 **Example**

3

4

5

6

7

8

9

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.

1 **Socket.BeginSendTo(System.Byte[],**
2 **System.Int32, System.Int32,**
3 **System.Net.Sockets.SocketFlags,**
4 **System.Net.EndPoint,**
5 **System.AsyncCallback, System.Object)**
6 **Method**

```
7 [ILASM]  
8 .method public hidebysig instance class System.IAsyncResult  
9 BeginSendTo(class System.Byte[] buffer, int32 offset, int32  
10 size, valuetype System.Net.Sockets.SocketFlags socketFlags,  
11 class System.Net.EndPoint remoteEP, class  
12 System.AsyncCallback callback, object state)  
  
13 [C#]  
14 public IAsyncResult BeginSendTo(byte[] buffer, int offset,  
15 int size, SocketFlags socketFlags, EndPoint remoteEP,  
16 AsyncCallback callback, object state)
```

17 **Summary**

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

20 **Parameters**

21
22

Parameter	Description
<i>buffer</i>	A System.Byte array storing data to send to the socket.
<i>offset</i>	A System.Int32 containing the zero-based position in <i>buffer</i> to begin sending data.
<i>size</i>	A System.Int32 containing the number of bytes to send.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	The System.Net.EndPoint associated with the socket to receive the data.
<i>callback</i>	A System.AsyncCallback delegate, or null .
<i>state</i>	An application-defined object, or null .

23
24
25

Return Value

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

3 Description

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

8
9 [Note: The **System.Net.Sockets.Socket.EndSendTo** method should
10 be called exactly once for each call to the
11 **System.Net.Sockets.Socket.BeginSendTo** method.]

12
13 If the *callback* parameter is not **null**, the method referenced by
14 *callback* is invoked when the asynchronous operation completes. The
15 **System.IAsyncResult** object returned by this method is passed as
16 the argument to the method referenced by *callback*. The method
17 referenced by *callback* can retrieve the results of the operation by
18 calling the **System.Net.Sockets.Socket.EndSendTo** method.

19
20 The *state* parameter can be any object that the caller wishes to have
21 available for the duration of the asynchronous operation. This object is
22 available via the **System.IAsyncResult.AsyncState** property of the
23 object returned by this method.

24
25 [Note: For more information, see
26 **System.Net.Sockets.Socket.SendTo**, the synchronous version of
27 this method.]

28 Exceptions

29
30

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

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

1
2
3

Example

4
5
6
7

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.

8
9

Permissions

10

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

11
12
13

1 Socket.Bind(System.Net.EndPoint)

2 Method

```
3 [ILASM]  
4 .method public hidebysig instance void Bind(class  
5 System.Net.EndPoint localEP)  
  
6 [C#]  
7 public void Bind(EndPoint localEP)
```

8 Summary

9 Associates the current instance with a local endpoint.

10 Parameters

11
12

Parameter	Description
<i>localEP</i>	The local System.Net.EndPoint to be associated with the socket.

13
14

14 Description

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

17
18 [Note: For connection-oriented protocols, this method is generally
19 used only on the server-side and is required to be called before the
20 first call to the **System.Net.Sockets.Socket.Listen** method. On the
21 client-side, binding is usually performed implicitly by the
22 **System.Net.Sockets.Socket.Connect** method.

23
24 For connectionless protocols, the
25 **System.Net.Sockets.Socket.Connect**
26 **System.Net.Sockets.Socket.SendTo**, and
27 **System.Net.Sockets.Socket.BeginSendTo** methods bind the
28 current instance to the local endpoint if the current instance has not
29 previously been bound.]

30 Exceptions

31
32

Exception	Condition
System.ArgumentNullException	<i>localEP</i> 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.

1
2
3
4

Permissions

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

5
6
7

1 Socket.Close() Method

```
2 [ILASM]  
3 .method public hidebysig instance void Close()  
4 [C#]  
5 public void Close()
```

6 Summary

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

9 Description

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

14
15 The socket attempts to perform a graceful closure when the
16 **System.Net.Sockets.SocketOptionName.Linger** socket option is
17 enabled and set to a non-zero linger time. In all other cases, closure is
18 forced and any pending data is lost.

19

1 Socket.Connect(System.Net.EndPoint)

2 Method

```
3 [ILASM]
4 .method public hidebysig instance void Connect(class
5 System.Net.EndPoint remoteEP)
6
7 [C#]
8 public void Connect(EndPoint remoteEP)
```

8 Summary

9 Associates the current instance with a remote endpoint.

10 Parameters

11
12

Parameter	Description
<i>remoteEP</i>	The System.Net.EndPoint associated with the socket to connect to.

13
14

14 Description

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

17
18 [Note: For connection-oriented protocols, this method establishes a
19 connection between the current instance and the socket associated
20 with *remoteEP*. This method is used only on the client-side. The
21 **System.Net.Sockets.Socket.Accept** method establishes the
22 connection on the server-side. Once the connection has been made,
23 data can be sent using the **System.Net.Sockets.Socket.Send**
24 method, and received using the
25 **System.Net.Sockets.Socket.Receive** method.

26
27 For connectionless protocols, the
28 **System.Net.Sockets.Socket.Connect** method can be used from
29 both client and server-sides, allowing the use of the
30 **System.Net.Sockets.Socket.Send** method instead of the
31 **System.Net.Sockets.Socket.SendTo** method. The
32 **System.Net.Sockets.Socket.RemoteEndPoint** property is set to
33 *remoteEP* and the **System.Net.Sockets.Socket.LocalEndPoint**
34 property is set to a value determined by the protocol; however, a
35 connection is not established. Subsequent data is required to be
36 received on the endpoint set in the
37 **System.Net.Sockets.Socket.LocalEndPoint** property.]

1 **Exceptions**

2

3

Exception	Condition
System.ArgumentNullException	<i>remoteEP</i> 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. [<i>Note</i> : 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.

4

5 **Permissions**

6

7

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

8

9

10

1 Socket.Dispose(System.Boolean) Method

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

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

7 Summary

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

11 Parameters

12 Parameter	13 Description
<i>disposing</i>	A System.Boolean . Specify true to release both managed and unmanaged resources; false to release only unmanaged resources.

16 Behaviors

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

22 Default

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

25 How and When to Override

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

29 Usage

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

31

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
<i>asyncResult</i>	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	<i>asyncResult</i> is null .
System.ArgumentException	<i>asyncResult</i> 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.

1
2
3
4
5
6
7
8
9

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.

1 Socket.EndConnect(System.IAsyncResult) 2 Method

```
3 [ILASM]  
4 .method public hidebysig instance void EndConnect(class  
5 System.IAsyncResult asyncResult)  
  
6 [C#]  
7 public void EndConnect(IAsyncResult asyncResult)
```

8 Summary

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

11 Parameters

12
13

Parameter	Description
<i>asyncResult</i>	A System.IAsyncResult object that holds the state information for the asynchronous operation.

14
15

15 Description

16 This method blocks if the asynchronous operation has not completed.

17
18
19
20
21
22
23
24
25

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.

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

31 Exceptions

32
33

Exception	Condition
System.ArgumentNullException	<i>asyncResult</i> is null .
System.ArgumentException	<i>asyncResult</i> 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. [<i>Note:</i> For additional information on causes of the SocketException , see the System.Net.Sockets.SocketException class.]
System.ObjectDisposedException	The current instance has been disposed.

1
2
3
4
5
6
7
8
9

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.

1 Socket.EndReceive(System.IAsyncResult) 2 Method

```
3 [ILASM]  
4 .method public hidebysig instance int32 EndReceive(class  
5 System.IAsyncResult asyncResult)  
6 [C#]  
7 public int EndReceive(IAsyncResult asyncResult)
```

8 Summary

9 Ends an asynchronous call to receive data from a socket.

10 Parameters

11
12

Parameter	Description
<i>asyncResult</i>	A System.IAsyncResult object that holds the state information for the asynchronous operation.

13
14
15

Return Value

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

17 Description

18 This method blocks if the asynchronous operation has not completed.

19

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

27

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

33 Exceptions

34
35

Exception	Condition
-----------	-----------

System.ArgumentNullException	<i>asyncResult</i> is null .
System.ArgumentException	<i>asyncResult</i> 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.

1
2
3
4
5
6
7
8
9

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
<i>asyncResult</i>	A System.IAsyncResult object that holds the state information for the asynchronous operation.
<i>endPoint</i>	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.

1 **Exceptions**

2

3

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

4

5

6

Example

7

8

9

10

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.

11

1 Socket.EndSend(System.IAsyncResult) 2 Method

```
3 [ILASM]  
4 .method public hidebysig instance int32 EndSend(class  
5 System.IAsyncResult asyncResult)  
  
6 [C#]  
7 public int EndSend(IAsyncResult asyncResult)
```

8 Summary

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

10 Parameters

11
12

Parameter	Description
<i>asyncResult</i>	A System.IAsyncResult object that holds the state information for the asynchronous operation.

13
14
15

Return Value

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

17 Description

18 This method blocks if the asynchronous operation has not completed.

19

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

27

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

33 Exceptions

34
35

Exception	Condition
-----------	-----------

System.ArgumentNullException	<i>asyncResult</i> is null .
System.ArgumentException	<i>asyncResult</i> 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.

1

2

Example

3

4

5

6

7

8

9

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.

1 Socket.EndSendTo(System.IAsyncResult) 2 Method

```
3 [ILASM]  
4 .method public hidebysig instance int32 EndSendTo(class  
5 System.IAsyncResult asyncResult)  
  
6 [C#]  
7 public int EndSendTo(IAsyncResult asyncResult)
```

8 Summary

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

11 Parameters

12
13

Parameter	Description
<i>asyncResult</i>	A System.IAsyncResult object that holds the state information for the asynchronous operation.

14
15
16

15 Return Value

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

18 Description

19 This method blocks if the asynchronous operation has not completed.

20

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

28

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

34 Exceptions

35
36

Exception	Condition
System.ArgumentNullException	<i>asyncResult</i> is null .
System.ArgumentException	<i>asyncResult</i> 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.

1
2
3
4
5
6
7
8

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.

1 Socket.Finalize() Method

```
2 [ILASM]  
3 .method family hidebysig virtual void Finalize()  
4 [C#]  
5 ~Socket()
```

6 Summary

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

9 Description

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

14
15 This method calls
16 **System.Net.Sockets.NetworkStream.Dispose(false)** to free
17 unmanaged resources used by the current instance.

18
19 This method overrides **System.Object.Finalize.**]

20

1 Socket.GetHashCode() Method

```
2 [ILASM]  
3 .method public hidebysig virtual int32 GetHashCode()  
4  
5 [C#]  
6 public override int GetHashCode()
```

6 Summary

7 Generates a hash code for the current instance.

8 Return Value

9

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

11 Description

12 The algorithm used to generate the hash code is unspecified.

13

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

15

1 **Socket.GetSocketOption(System.Net.Sockets.SocketOptionLevel,**
2 **System.Net.Sockets.SocketOptionName)**
3 **Method**
4

```
5 [ILASM]  
6 .method public hidebysig instance object  
7 GetSocketOption(valuetype  
8 System.Net.Sockets.SocketOptionLevel optionLevel, valuetype  
9 System.Net.Sockets.SocketOptionName optionName)  
  
10 [C#]  
11 public object GetSocketOption(SocketOptionLevel  
12 optionLevel, SocketOptionName optionName)
```

13 **Summary**

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

15 **Parameters**

16
17

Parameter	Description
<i>optionLevel</i>	One of the values defined in the System.Net.Sockets.SocketOptionLevel enumeration.
<i>optionName</i>	One of the values defined in the System.Net.Sockets.SocketOptionName enumeration.

18
19
20

19 **Return Value**

21 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.

1

2 Description

3 Socket options determine the behavior of the current instance.

4

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

10 Exceptions

11

12

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

13

14 Example

15

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

19

20

[C#]

21

```
using System;
```

22

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

23

```
class OptionTest{
```

24

```
    public static void Main() {
```

25

```
        // Get the current option values.
```

26

```
        Socket someSocket =
```

27

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

28

```
        LingerOption lingerOp =
```

29

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

30

31

32

33

34

35

36

37

38

```

1      int receiveBuffer =
2          (int)someSocket.GetSocketOption(
3              SocketOptionLevel.Socket,
4              SocketOptionName.ReceiveBuffer);
5
6      Console.WriteLine(
7          "Linger option is {0} and set to {1} seconds.",
8          lingerOp.Enabled.ToString(),
9          lingerOp.LingerTime.ToString());
10
11     Console.WriteLine(
12         "Size of the receive buffer is {0} bytes.",
13         receiveBuffer.ToString());
14
15     // Change the options.
16     lingerOp = new LingerOption(true, 10);
17     someSocket.SetSocketOption(
18         SocketOptionLevel.Socket,
19         SocketOptionName.Linger,
20         lingerOp);
21
22     someSocket.SetSocketOption(
23         SocketOptionLevel.Socket,
24         SocketOptionName.ReceiveBuffer,
25         2048);
26
27     Console.WriteLine(
28         "The SetSocketOption method has been called.");
29
30     // Get the new option values.
31     lingerOp =
32         (LingerOption)someSocket.GetSocketOption(
33             SocketOptionLevel.Socket,
34             SocketOptionName.Linger);
35
36     receiveBuffer =
37         (int)someSocket.GetSocketOption(
38             SocketOptionLevel.Socket,
39             SocketOptionName.ReceiveBuffer);
40
41     Console.WriteLine(
42         "Linger option is now {0} and set to {1} seconds.",
43         lingerOp.Enabled.ToString(),
44         lingerOp.LingerTime.ToString());
45
46     Console.WriteLine(
47         "Size of the receive buffer is now {0} bytes.",
48         receiveBuffer.ToString());
49 }
50 }
51

```

52 The output is
53

1 Linger option is False and set to 0 seconds.
2
3
4 Size of the receive buffer is 8192 bytes.
5
6
7 The SetSocketOption method has been called.
8
9
10 Linger option is now True and set to 10 seconds.
11
12
13 Size of the receive buffer is now 2048 bytes.
14
15

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
<i>optionLevel</i>	One of the values defined in the System.Net.Sockets.SocketOptionLevel enumeration.
<i>optionName</i>	One of the values defined in the System.Net.Sockets.SocketOptionName enumeration.
<i>optionValue</i>	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
-----------	-----------

1
2
3

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

1 Socket.GetSocketOption(System.Net.Sockets.SocketOptionLevel, 2 3 System.Net.Sockets.SocketOptionName, 4 System.Int32) Method

```
5 [ILASM]  
6 .method public hidebysig instance class System.Byte[]  
7 GetSocketOption(valuetype  
8 System.Net.Sockets.SocketOptionLevel optionLevel, valuetype  
9 System.Net.Sockets.SocketOptionName optionName, int32  
10 optionLength)  
11  
12 [C#]  
13 public byte[] GetSocketOption(SocketOptionLevel  
optionLevel, SocketOptionName optionName, int optionLength)
```

14 Summary

15 Retrieves the value of the specified socket option.

16 Parameters

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

19 Return Value

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

24 Description

25 Socket options determine the behavior of the current instance.

26
27 The *optionLength* parameter is used to allocate an array to store the
28 value of the specified option. When this value is smaller than the
29 number of bytes required to store the value of the specified option, a
30 **System.Net.Sockets.SocketException** exception is thrown. When
31 this value is greater than or equal to the number of bytes required to

1 store the value of the specified option, the array returned by this
2 method is allocated to be exactly the required length.

3 **Exceptions**

4
5

Exception	Condition
System.Net.Sockets.SocketException	<i>optionLength</i> 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. [<i>Note:</i> For additional information on causes of the SocketException , see the System.Net.Sockets.SocketException class.]
System.ObjectDisposedException	The current instance has been disposed.

6
7
8

1 Socket.IOControl(System.Int32, 2 System.Byte[], System.Byte[]) Method

```
3 [ILASM]  
4 .method public hidebysig instance int32 IOControl(int32  
5 ioControlCode, class System.Byte[] optionInValue, class  
6 System.Byte[] optionOutValue)  
  
7 [C#]  
8 public int IOControl(int ioControlCode, byte[]  
9 optionInValue, byte[] optionOutValue)
```

10 Summary

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

13 Parameters

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

16 Return Value

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

21 Description

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

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

28 [Note: Input data is not required for all control codes. Output data is
29 not supplied by all control codes and, if not supplied, the return value
30 is 0.]
31
32

1 Exceptions

2

3

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.

4

5

Example

6

7

8

9

10

11

12

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#]

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

```
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};
```

```

1      byte[]outValue = {0x00, 0x00, 0x00, 0x00};
2
3      someSocket.IOControl(fionRead, inValue, outValue);
4
5      uint bytesAvail = BitConverter.ToUInt32(outValue, 0);
6
7      Console.WriteLine(
8          "There are {0} bytes available to be read.",
9          bytesAvail.ToString());
10     }
11 }
12

```

13 The output is

14
15 There are 0 bytes available to be read.

16

17 **Permissions**

18

19

Permission	Description
System.Security.Permissions.SecurityPermission	Requires permission to access unmanaged code. See System.Security.Permissions.SecurityPermissionFlagUnmanagedCode .

20

21

22

1 Socket.Listen(System.Int32) Method

```
2 [ILASM]  
3 .method public hidebysig instance void Listen(int32  
4 backlog)  
5 [C#]  
6 public void Listen(int backlog)
```

7 Summary

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

10 Parameters

Parameter	Description
<i>backlog</i>	A System.Int32 containing the maximum length of the queue of pending connections.

14 Description

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

20
21 If a connection request arrives and the queue is full, a
22 **System.Net.Sockets.SocketException** is thrown on the client.

23
24 A socket in the listening state has no remote endpoint associated with
25 it. Attempting to access the
26 **System.Net.Sockets.Socket.RemoteEndPoint** property throws a
27 **System.Net.Sockets.SocketException** exception.

28
29 This method is ignored if called more than once on the current
30 instance.

31
32 [Note: This method is used only on the server-side of connection-
33 oriented protocols. Call the **System.Net.Sockets.Socket.Bind**
34 method before this method is called the first time. Call the
35 **System.Net.Sockets.Socket.Listen** method before the first call to
36 the **System.Net.Sockets.Socket.Accept** method.]

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

1
2
3

1 Socket.Poll(System.Int32, 2 System.Net.Sockets.SelectMode) Method

```
3 [ILASM]  
4 .method public hidebysig instance bool Poll(int32  
5 microSeconds, valuetype System.Net.Sockets.SelectMode mode)  
  
6 [C#]  
7 public bool Poll(int microSeconds, SelectMode mode)
```

8 Summary

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

10 Parameters

11
12

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

13
14
15

14 Return Value

16 A **System.Boolean** where **true** indicates the current instance satisfies
17 at least one of the conditions in the following table corresponding to
18 the specified **System.Net.Sockets.SelectMode** value; otherwise,
19 **false**. **false** is returned if the status of the current instance cannot be
20 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.

1

2 **Exceptions**

3

4

Exception	Condition
System.NotSupportedException	<i>mode</i> 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. [<i>Note:</i> For additional information on causes of the SocketException , see the System.Net.Sockets.SocketException class.]
System.ObjectDisposedException	The current instance has been disposed.

5

6

7

1 Socket.Receive(System.Byte[], 2 System.Int32, 3 System.Net.Sockets.SocketFlags) Method

```
4 [ILASM]  
5 .method public hidebysig instance int32 Receive(class  
6 System.Byte[] buffer, int32 size, valuetype  
7 System.Net.Sockets.SocketFlags socketFlags)  
  
8 [C#]  
9 public int Receive(byte[] buffer, int size, SocketFlags  
10 socketFlags)
```

11 Summary

12 Receives data from a socket.

13 Parameters

14
15

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>size</i>	A System.Int32 containing the number of bytes to receive.
<i>socketFlags</i>	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 .

16
17
18

17 Return Value

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

20 Description

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

23 Exceptions

24
25

Exception	Condition
System.ArgumentNullException	<i>buffer</i> is null .
System.ArgumentOutOfRangeException	<i>size</i> < 0.

	-or- <i>size > buffer.Length.</i>
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	<i>socketFlags</i> is not a valid combination of values. -or- An error occurred while accessing the socket. [<i>Note:</i> 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.

1
2
3
4

Permissions

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

5
6
7

1 Socket.Receive(System.Byte[], 2 System.Net.Sockets.SocketFlags) Method

```
3 [ILASM]  
4 .method public hidebysig instance int32 Receive(class  
5 System.Byte[] buffer, valuetype  
6 System.Net.Sockets.SocketFlags socketFlags)  
7  
8 [C#]  
9 public int Receive(byte[] buffer, SocketFlags socketFlags)
```

9 Summary

10 Receives data from a socket.

11 Parameters

12
13

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>socketFlags</i>	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 .

14
15
16

15 Return Value

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

18 Description

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

22 Exceptions

23
24

Exception	Condition
System.ArgumentNullException	<i>buffer</i> is null .
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	<i>socketFlags</i> 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.

1
2
3
4

Permissions

Permission	Description
System.Net.SocketPermission	Requires permission to accept connections. [<i>Note:</i> See System.Net.NetworkAccess.Accept.]

5
6
7

1 Socket.Receive(System.Byte[]) Method

```
2 [ILASM]  
3 .method public hidebysig instance int32 Receive(class  
4 System.Byte[] buffer)  
  
5 [C#]  
6 public int Receive(byte[] buffer)
```

7 Summary

8 Receives data from a socket.

9 Parameters

10
11

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.

12
13
14

Return Value

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

16 Description

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

20 Exceptions

21
22

Exception	Condition
System.ArgumentNullException	<i>buffer</i> 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.

1
2
3
4

Permissions

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

5
6
7

1 Socket.Receive(System.Byte[], 2 System.Int32, System.Int32, 3 System.Net.Sockets.SocketFlags) Method

```
4 [ILASM]  
5 .method public hidebysig instance int32 Receive(class  
6 System.Byte[] buffer, int32 offset, int32 size, valuetype  
7 System.Net.Sockets.SocketFlags socketFlags)  
  
8 [C#]  
9 public int Receive(byte[] buffer, int offset, int size,  
10 SocketFlags socketFlags)
```

11 Summary

12 Receives data from a socket.

13 Parameters

14
15

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>offset</i>	A System.Int32 containing the zero-based position in <i>buffer</i> to begin storing the received data.
<i>size</i>	A System.Int32 containing the number of bytes to receive.
<i>socketFlags</i>	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 .

16
17
18

17 Return Value

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

20 Description

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

23
24
25
26
27
28
29

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.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

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.

20 **Exceptions**
21
22

Exception	Condition
System.ArgumentNullException	<i>buffer</i> is null .
System.ArgumentOutOfRangeException	<i>offset</i> < 0. -or- <i>offset</i> > <i>buffer.Length</i> . -or- <i>size</i> < 0. -or- <i>size</i> > <i>buffer.Length</i> - <i>offset</i> .
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	<i>socketFlags</i> 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. [<i>Note:</i> 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.

1
2
3
4

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 .

5
6
7

1 Socket.ReceiveFrom(System.Byte[], 2 System.Int32, System.Int32, 3 System.Net.Sockets.SocketFlags, 4 System.Net.EndPoint&) Method

```
5 [ILASM]  
6 .method public hidebysig instance int32 ReceiveFrom(class  
7 System.Byte[] buffer, int32 offset, int32 size, valuetype  
8 System.Net.Sockets.SocketFlags socketFlags, class  
9 System.Net.EndPoint& remoteEP)  
  
10 [C#]  
11 public int ReceiveFrom(byte[] buffer, int offset, int size,  
12 SocketFlags socketFlags, ref EndPoint remoteEP)
```

13 Summary

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

16 Parameters

17
18

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>offset</i>	A System.Int32 containing the zero-based position in <i>buffer</i> to begin storing the received data.
<i>size</i>	A System.Int32 containing the number of bytes to receive.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	A reference to the System.Net.EndPoint associated with the socket that sent the data.

19
20
21

Return Value

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

23 Description

24 For connectionless protocols, when this method successfully
25 completes, *remoteEP* contains the endpoint associated with the socket

1 that sent the data.

2
3 For connection-oriented protocols, *remoteEP* is left unchanged.

4
5 The **System.Net.Sockets.Socket.LocalEndPoint** property is
6 required to be set before this method is called or a
7 **System.Net.Sockets.SocketException** is thrown.

8
9 The **System.Net.Sockets.Socket.Blocking** property of the socket
10 determines the behavior of this method when no incoming data is
11 available. When **false**, the **System.Net.Sockets.SocketException**
12 exception is thrown. When **true**, this method blocks and waits for data
13 to arrive.

14
15 For **System.Net.Sockets.SocketType.Stream** socket types, if the
16 remote socket was shut down gracefully, and all data was received,
17 this method immediately returns zero, regardless of the blocking state.

18
19 For message-oriented sockets, if the message is larger than the size of
20 *buffer*, the buffer is filled with the first part of the message, and the
21 **System.Net.Sockets.SocketException** exception is thrown. For
22 unreliable protocols, the excess data is lost; for reliable protocols, the
23 data is retained by the service provider.

24
25 When the **System.Net.Sockets.SocketFlags.OutOfBand** flag is
26 specified as part of the *socketFlags* parameter and the socket is
27 configured for in-line reception of out-of-band (OOB) data (using the
28 **System.Net.Sockets.SocketOptionName.OutOfBandInline** socket
29 option) and OOB data is available, only OOB data is returned.

30
31 When the **System.Net.Sockets.SocketFlags.Peek** flag is specified
32 as part of the *socketFlags* parameter, available data is copied into
33 *buffer* but is not removed from the system buffer.

34 Exceptions

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

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

1
2
3
4

Permissions

Permission	Description
System.Net.SocketPermission	<p>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.</p> <p>Requires permission to make a connection to the endpoint defined by <i>remoteEP</i>. See System.Net.NetworkAccess.Connect.</p>

5
6
7

1 Socket.ReceiveFrom(System.Byte[], 2 System.Int32, 3 System.Net.Sockets.SocketFlags, 4 System.Net.EndPoint&) Method

```
5 [ILASM]  
6 .method public hidebysig instance int32 ReceiveFrom(class  
7 System.Byte[] buffer, int32 size, valuetype  
8 System.Net.Sockets.SocketFlags socketFlags, class  
9 System.Net.EndPoint& remoteEP)  
  
10 [C#]  
11 public int ReceiveFrom(byte[] buffer, int size, SocketFlags  
12 socketFlags, ref EndPoint remoteEP)
```

13 Summary

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

16 Parameters

17
18

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>size</i>	A System.Int32 containing the number of bytes to receive.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	A reference to the System.Net.EndPoint associated with the socket that sent the data.

19
20
21

Return Value

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

23 Description

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

1 **Exceptions**

2
3

Exception	Condition
System.ArgumentNullException	<i>buffer</i> or <i>remoteEP</i> is null .
System.ArgumentOutOfRangeException	<i>size</i> < 0. -or- <i>size</i> > <i>buffer.Length</i> .
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	<i>socketFlags</i> 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.

4
5
6
7

Permissions

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

8
9
10

1 Socket.ReceiveFrom(System.Byte[], 2 System.Net.Sockets.SocketFlags, 3 System.Net.EndPoint&) Method

```
4 [ILASM]  
5 .method public hidebysig instance int32 ReceiveFrom(class  
6 System.Byte[] buffer, valuetype  
7 System.Net.Sockets.SocketFlags socketFlags, class  
8 System.Net.EndPoint& remoteEP)  
  
9 [C#]  
10 public int ReceiveFrom(byte[] buffer, SocketFlags  
11 socketFlags, ref EndPoint remoteEP)
```

12 Summary

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

15 Parameters

16
17

Parameter	Description
<i>buffer</i>	A System.Byte array to store data received from the socket.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	A reference to the System.Net.EndPoint associated with the socket that sent the data.

18
19
20

19 Return Value

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

22 Description

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

26 Exceptions

27
28

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

1
2
3
4

Permissions

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

5
6
7

1 Socket.ReceiveFrom(System.Byte[], 2 System.Net.EndPoint&) Method

```
3 [ILASM]  
4 .method public hidebysig instance int32 ReceiveFrom(class  
5 System.Byte[] buffer, class System.Net.EndPoint& remoteEP)  
  
6 [C#]  
7 public int ReceiveFrom(byte[] buffer, ref EndPoint  
8 remoteEP)
```

9 Summary

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

12 Parameters

13
14

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

15
16
17

16 Return Value

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

19 Description

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

23 Exceptions

24
25

Exception	Condition
System.ArgumentNullException	<i>buffer</i> or <i>remoteEP</i> 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.

1
2
3
4

Permissions

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

5
6
7

1 **Socket.Select(System.Collections.IList,**
 2 **System.Collections.IList,**
 3 **System.Collections.IList, System.Int32)**
 4 **Method**

```
5 [ILASM]
6 .method public hidebysig static void Select(class
7 System.Collections.IList checkRead, class
8 System.Collections.IList checkWrite, class
9 System.Collections.IList checkError, int32 microseconds)

10 [C#]
11 public static void Select(IList checkRead, IList
12 checkWrite, IList checkError, int microseconds)
```

13 **Summary**

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

16 **Parameters**

17
 18

Parameter	Description
<i>checkRead</i>	A System.Collections.IList object containing the System.Net.Sockets.Socket instances to check for read status.
<i>checkWrite</i>	A System.Collections.IList object containing the System.Net.Sockets.Socket instances to check for write status.
<i>checkError</i>	A System.Collections.IList object containing the System.Net.Sockets.Socket instances to check for error status.
<i>microSeconds</i>	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.

19
 20

Description

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

List	Condition to remain in list
<i>checkRead</i>	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

	<p>is set).</p> <p>-or-</p> <p>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.</p> <p>-or-</p> <p>The connection has been closed, reset, or terminated.</p>
<i>checkWrite</i>	<p>Data can be sent.</p> <p>-or-</p> <p>A non-blocking System.Net.Sockets.Socket.Connect method is being processed and the connection has succeeded.</p>
<i>checkError</i>	<p>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.</p> <p>-or-</p> <p>A non-blocking System.Net.Sockets.Socket.Connect method is being processed and the connection has failed.</p>

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

[*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: <i>checkRead</i> , <i>checkWrite</i> , and <i>checkError</i> .

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(
```

```
1         "socket3 is {0}in writeList.",
2         writeList.Contains(socket3) ? "": "not ");
3     Console.WriteLine(
4         "socket3 is {0}in errorList.",
5         errorList.Contains(socket3) ? "": "not ");
6
7     Socket.Select(readList, writeList, errorList, 10);
8     Console.WriteLine("The Select method has been
9 called.");
10    Console.WriteLine(
11        "socket3 is {0}in readList.",
12        readList.Contains(socket3) ? "": "not ");
13    Console.WriteLine(
14        "socket3 is {0}in writeList.",
15        writeList.Contains(socket3) ? "": "not ");
16    Console.WriteLine(
17        "socket3 is {0}in errorList.",
18        errorList.Contains(socket3) ? "": "not ");
19    }
20 }
21
```

```
22     The output is
23
24     socket3 is placed in readList and errorList.
25
26
27     socket3 is in readList.
28
29
30     socket3 is not in writeList.
31
32
33     socket3 is in errorList.
34
35
36     The Select method has been called.
37
```

```
1
2     socket3 is not in readList.
3
4
5     socket3 is not in writeList.
6
7
8     socket3 is not in errorList.
9
10
```

1 Socket.Send(System.Byte[], 2 System.Int32, 3 System.Net.Sockets.SocketFlags) Method

```
4 [ILASM]  
5 .method public hidebysig instance int32 Send(class  
6 System.Byte[] buffer, int32 size, valuetype  
7 System.Net.Sockets.SocketFlags socketFlags)  
  
8 [C#]  
9 public int Send(byte[] buffer, int size, SocketFlags  
10 socketFlags)
```

11 Summary

12 Sends data to a connected socket.

13 Parameters

14
15

Parameter	Description
<i>buffer</i>	A System.Byte array containing data to send to the socket.
<i>size</i>	A System.Int32 containing the number of bytes to send.
<i>socketFlags</i>	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 .

16
17
18

17 Return Value

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

20 Description

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

23 Exceptions

24
25

Exception	Condition
System.ArgumentNullException	<i>buffer</i> is null .
System.ArgumentOutOfRangeException	<i>size</i> < 0.

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

1
2
3

1 Socket.Send(System.Byte[], 2 System.Net.Sockets.SocketFlags) Method

```
3 [ILASM  
4 .method public hidebysig instance int32 Send(class  
5 System.Byte[] buffer, valuetype  
6 System.Net.Sockets.SocketFlags socketFlags)  
7  
8 [C#  
9 public int Send(byte[] buffer, SocketFlags socketFlags)
```

9 Summary

10 Sends data to a connected socket.

11 Parameters

12
13

Parameter	Description
<i>buffer</i>	A System.Byte array containing data to send to the socket.
<i>socketFlags</i>	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 .

14
15
16

15 Return Value

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

18 Description

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

22 Exceptions

23
24

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

1
2
3

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

1 Socket.Send(System.Byte[]) Method

```
2 [ILASM]
3 .method public hidebysig instance int32 Send(class
4 System.Byte[] buffer)
5
6 [C#]
7 public int Send(byte[] buffer)
```

7 Summary

8 Sends data to a connected socket.

9 Parameters

10
11

Parameter	Description
<i>buffer</i>	A System.Byte array containing data to send to the socket.

12
13
14

13 Return Value

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

16 Description

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

20 Exceptions

21
22

Exception	Condition
System.ArgumentNullException	<i>buffer</i> 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.

1
2
3

1 Socket.Send(System.Byte[], 2 System.Int32, System.Int32, 3 System.Net.Sockets.SocketFlags) Method

```
4 [ILASM]  
5 .method public hidebysig instance int32 Send(class  
6 System.Byte[] buffer, int32 offset, int32 size, valuetype  
7 System.Net.Sockets.SocketFlags socketFlags)  
  
8 [C#]  
9 public int Send(byte[] buffer, int offset, int size,  
10 SocketFlags socketFlags)
```

11 Summary

12 Sends data to a connected socket.

13 Parameters

14
15

Parameter	Description
<i>buffer</i>	A System.Byte array containing data to send to the socket.
<i>offset</i>	A System.Int32 that specifies the zero-based position in buffer that is the starting location of the data to send.
<i>size</i>	A System.Int32 containing the number of bytes to send.
<i>socketFlags</i>	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 .

16
17
18

17 Return Value

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

20 Description

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

24
25 For connectionless protocols, calling the
26 **System.Net.Sockets.Socket.Connect** methods sets the
27 **System.Net.Sockets.Socket.RemoteEndPoint** property and allows
28 the **System.Net.Sockets.Socket.Send** method to be used instead of
29 the **System.Net.Sockets.Socket.SendTo** method.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

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.

17 **Exceptions**
18
19

Exception	Condition
System.ArgumentNullException	<i>buffer</i> is null .
System.ArgumentOutOfRangeException	<i>offset</i> < 0. -or- <i>offset</i> > <i>buffer.Length</i> . -or- <i>size</i> < 0. -or- <i>size</i> > <i>buffer.Length</i> - <i>offset</i> .
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	<i>socketFlags</i> 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

1
2
3

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

1 Socket.SendTo(System.Byte[], 2 System.Int32, System.Int32, 3 System.Net.Sockets.SocketFlags, 4 System.Net.EndPoint) Method

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

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

13 Summary

14 Sends data to the socket associated with the specified endpoint.

15 Parameters

Parameter	Description
<i>buffer</i>	A System.Byte array containing data to send to the socket.
<i>offset</i>	A System.Int32 that specifies the zero-based position in buffer that is the starting location of the data to send.
<i>size</i>	A System.Int32 containing the number of bytes to send.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	The System.Net.EndPoint associated with the socket to receive the data.

18 Return Value

19 A **System.Int32** containing the number of bytes sent.
20

22 Description

23 For connection-oriented protocols and connected sockets using
24 connectionless protocols, *remoteEP* overrides the endpoint specified in

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

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

	-or- An error occurred while accessing the socket. [<i>Note:</i> 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.

1
2
3
4

Permissions

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

5
6
7

1 Socket.SendTo(System.Byte[], 2 System.Int32, 3 System.Net.Sockets.SocketFlags, 4 System.Net.EndPoint) Method

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

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

13 Summary

14 Sends data to the socket associated with the specified endpoint.

15 Parameters

16
17

Parameter	Description
<i>buffer</i>	A System.Byte array containing data to send to the socket.
<i>size</i>	A System.Int32 containing the number of bytes to send.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	The System.Net.EndPoint associated with the socket to receive the data.

18
19
20

19 Return Value

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

22 Description

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

1 **Exceptions**

2
3

Exception	Condition
System.ArgumentNullException	<i>buffer</i> or <i>remoteEP</i> is null .
System.ArgumentOutOfRangeException	<i>size</i> < 0. -or- <i>size</i> > <i>buffer.Length</i> .
System.InvalidOperationException	An asynchronous call is pending and a blocking method has been called.
System.Net.Sockets.SocketException	<i>socketFlags</i> 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.

4
5
6
7

Permissions

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

8
9
10

1 Socket.SendTo(System.Byte[], 2 System.Net.Sockets.SocketFlags, 3 System.Net.EndPoint) Method

```
4 [ILASM]  
5 .method public hidebysig instance int32 SendTo(class  
6 System.Byte[] buffer, valuetype  
7 System.Net.Sockets.SocketFlags socketFlags, class  
8 System.Net.EndPoint remoteEP)  
  
9 [C#]  
10 public int SendTo(byte[] buffer, SocketFlags socketFlags,  
11 EndPoint remoteEP)
```

12 Summary

13 Sends data to the socket associated with the specified endpoint.

14 Parameters

15
16

Parameter	Description
<i>buffer</i>	A System.Byte array containing data to send to the socket.
<i>socketFlags</i>	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 .
<i>remoteEP</i>	The System.Net.EndPoint associated with the socket to receive the data.

17
18
19

18 Return Value

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

21 Description

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

25 Exceptions

26
27

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

1
2
3
4

Permissions

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

5
6
7

1 Socket.SendTo(System.Byte[], 2 System.Net.EndPoint) Method

```
3 [ILASM]  
4 .method public hidebysig instance int32 SendTo(class  
5 System.Byte[] buffer, class System.Net.EndPoint remoteEP)  
  
6 [C#]  
7 public int SendTo(byte[] buffer, EndPoint remoteEP)
```

8 Summary

9 Sends data to the socket associated with the specified endpoint.

10 Parameters

11
12

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

13
14
15

14 Return Value

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

17 Description

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

21 Exceptions

22
23

Exception	Condition
System.ArgumentNullException	<i>buffer</i> or <i>remoteEP</i> 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.

1
2
3
4

Permissions

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

5
6
7

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
<i>optionLevel</i>	One of the values defined in the System.Net.Sockets.SocketOptionLevel enumeration.
<i>optionName</i>	One of the values defined in the System.Net.Sockets.SocketOptionName enumeration.
<i>optionValue</i>	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

1
2
3
4
5
6
7

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.
------	--

1
2
3
4
5
6

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.

7
8
9
10
11
12
13
14

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.
---------	---

1
2
3
4
5
6
7
8

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.

9
10
11
12
13
14
15
16
17

[*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.]

18 **Exceptions**
19
20

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [<i>Note:</i> 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.

21
22 **Permissions**
23
24

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

1
2
3

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
<i>optionLevel</i>	One of the values defined in the System.Net.Sockets.SocketOptionLevel enumeration.
<i>optionName</i>	One of the values defined in the System.Net.Sockets.SocketOptionName enumeration.
<i>optionValue</i>	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

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

4 **Exceptions**

5
6

Exception	Condition
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [<i>Note:</i> 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.

7

8 **Permissions**

9
10

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

11
12
13

1 Socket.SetSocketOption(System.Net.Sockets.SocketOptionLevel, 2 System.Net.Sockets.SocketOptionName, 3 System.Object) Method 4

```
5 [ILASM]  
6 .method public hidebysig instance void  
7 SetSocketOption(valuetype  
8 System.Net.Sockets.SocketOptionLevel optionLevel, valuetype  
9 System.Net.Sockets.SocketOptionName optionName, object  
10 optionValue)  
  
11 [C#]  
12 public void SetSocketOption(SocketOptionLevel optionLevel,  
13 SocketOptionName optionName, object optionValue)
```

14 Summary

15 Sets the
16 **System.Net.Sockets.SocketOptionName.AddMembership**,
17 **System.Net.Sockets.SocketOptionName.DropMembership**, or
18 **System.Net.Sockets.SocketOptionName.Linger** socket options.

19 Parameters

20
21

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

22

23 Description

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

27

28 The following table summarizes the valid combinations of input
29 parameters.

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

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

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

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.]

23
24
25

Exceptions

Exception	Condition
System.ArgumentException	<i>optionLevel</i> , <i>optionName</i> , or <i>optionValue</i> specified an invalid value.
System.ArgumentNullException	<i>optionValue</i> is null .
System.Net.Sockets.SocketException	An error occurred while accessing the socket. [<i>Note:</i> 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.

26
27

Permissions

1
2

Permission	Description
System.Security.Permissions.SecurityPermission	The System.Net.Sockets.SocketOptionName.AddMember and System.Net.Sockets.SocketOptionName.DropMember options require permission to access unmanaged code. See System.Security.Permissions.SecurityPermissionFlags.UnmanagedCode .

3
4
5

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

```
3 [ILASM]  
4 .method public hidebysig instance void Shutdown(valuetype  
5 System.Net.Sockets.SocketShutdown how)  
  
6 [C#]  
7 public void Shutdown(SocketShutdown how)
```

8 Summary

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

10 Parameters

11
12

Parameter	Description
<i>how</i>	One of the values defined in the System.Net.Sockets.SocketShutdown enumeration.

13
14

14 Description

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

20
21 When *how* is set to
22 **System.Net.Sockets.SocketShutdown.Receive**, the socket on the
23 other end of the connection is notified that the current instance will not
24 receive any more data. After all the data currently queued on the
25 current instance is received, any subsequent calls to the
26 **System.Net.Sockets.Socket.Receive** method cause a
27 **System.Net.Sockets.SocketException** exception to be thrown.

28
29 Setting *how* to **System.Net.Sockets.SocketShutdown.Both**
30 terminates both sends and receives as described above. Once this
31 occurs, the socket cannot be used.

32
33 [*Note:* To free resources allocated by the current instance, call the
34 **System.Net.Sockets.Socket.Close** method.

35
36 Expected common usage is for the
37 **System.Net.Sockets.Socket.Shutdown** method to be called before
38 the **System.Net.Sockets.Socket.Close** method to ensure that all
39 pending data is sent or received.]

1 **Exceptions**
2
3

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

4
5
6

1 Socket.System.IDisposable.Dispose() 2 Method

```
3 [ILASM]  
4 .method private final hidebysig virtual void  
5 System.IDisposable.Dispose()  
6  
7 [C#]  
8 void IDisposable.Dispose()
```

8 Summary

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

11

1 Socket.AddressFamily Property

```
2 [ILASM]
3 .property valuetype System.Net.Sockets.AddressFamily
4 AddressFamily { public hidebyref specialname instance
5 valuetype System.Net.Sockets.AddressFamily
6 get_AddressFamily() }
7
8 [C#]
9 public AddressFamily AddressFamily { get; }
```

9 Summary

10 Gets the address family of the current instance.

11 Property Value

12

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

15 Description

16 This property is read-only.

17

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

21

1 Socket.Available Property

```
2 [ILASM]
3 .property int32 Available { public hidebysig specialname
4 instance int32 get_Available() }
5
6 [C#]
7 public int Available { get; }
```

7 Summary

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

11 Property Value

12

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

15 Description

16 This property is read-only.

17

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

22

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

26 Exceptions

27

28

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

29

30

31

1 Socket.Blocking Property

```
2 [ILASM]
3 .property bool Blocking { public hidebysig specialname
4 instance bool get_Blocking() public hidebysig specialname
5 instance void set_Blocking(bool value) }
6
7 [C#]
8 public bool Blocking { get; set; }
```

8 Summary

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

11 Property Value

12

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

15 Description

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

18

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

22 Exceptions

23

24

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

25

26

27

1 Socket.Connected Property

```
2 [ILASM]
3 .property bool Connected { public hideby sig specialname
4 instance bool get_Connected() }
5 [C#]
6 public bool Connected { get; }
```

7 Summary

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

10 Property Value

11

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

15 Description

16 This property is read-only.

17

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

22

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

26

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

30

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

3 Socket.Handle Property

```
4 [ILASM]  
5 .property valuetype System.IntPtr Handle { public hidebysig  
6 specialname instance valuetype System.IntPtr get_Handle() }  
7 [C#]  
8 public IntPtr Handle { get; }
```

9 Summary

10 Gets the operating system handle for the current instance.

11 Property Value

12

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

15 Description

16 This property is read-only.

17 Permissions

18

19

Permission	Description
System.Security.Permissions.SecurityPermission	Requires permission to access unmanaged code. See System.Security.Permissions.SecurityPermissionFlagUnmanagedCode .

20

21

22

1 Socket.LocalEndPoint Property

```
2 [ILASM]
3 .property class System.Net.EndPoint LocalEndPoint { public
4 hidebysig specialname instance class System.Net.EndPoint
5 get_LocalEndPoint() }
6
7 [C#]
8 public EndPoint LocalEndPoint { get; }
```

8 Summary

9 Gets the local endpoint associated with the current instance.

10 Property Value

11

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

13 Description

14 This property is read-only.

15

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

18

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

25 Exceptions

26

27

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

28

29

30

1 Socket.ProtocolType Property

```
2 [ILASM]
3 .property valuetype System.Net.Sockets.ProtocolType
4 ProtocolType { public hidebysig specialname instance
5 valuetype System.Net.Sockets.ProtocolType
6 get_ProtocolType() }
7
8 [C#]
9 public ProtocolType ProtocolType { get; }
```

9 Summary

10 Gets the protocol type of the current instance.

11 Property Value

12

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

15 Description

16 This property is read-only.

17

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

21

1 Socket.RemoteEndPoint Property

```
2 [ILASM]
3 .property class System.Net.EndPoint RemoteEndPoint { public
4 hidebysig specialname instance class System.Net.EndPoint
5 get_RemoteEndPoint() }
6
7 [C#]
8 public EndPoint RemoteEndPoint { get; }
```

8 Summary

9 Gets the remote endpoint associated with the current instance.

10 Property Value

11

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

14 Description

15 This property is read-only.

16

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

19

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

24

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

28 Exceptions

29

30

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

31

32

33

1 Socket.SocketType Property

```
2 [ILASM]
3 .property valuetype System.Net.Sockets.SocketType
4 SocketType { public hidebysig specialname instance
5 valuetype System.Net.Sockets.SocketType get_SocketType() }
6
7 [C#]
8 public SocketType SocketType { get; }
```

8 Summary

9 Gets the socket type of the current instance.

10 Property Value

11

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

14 Description

15 This property is read-only.

16

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

18