University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

Network Programming 2
برمجة شبكات 2

Asst. Lectur. Atheer Raheem

م.م. اثير رحيم محسن

cs.uotechnology.ed

DomainNameSystemClass(DNS)

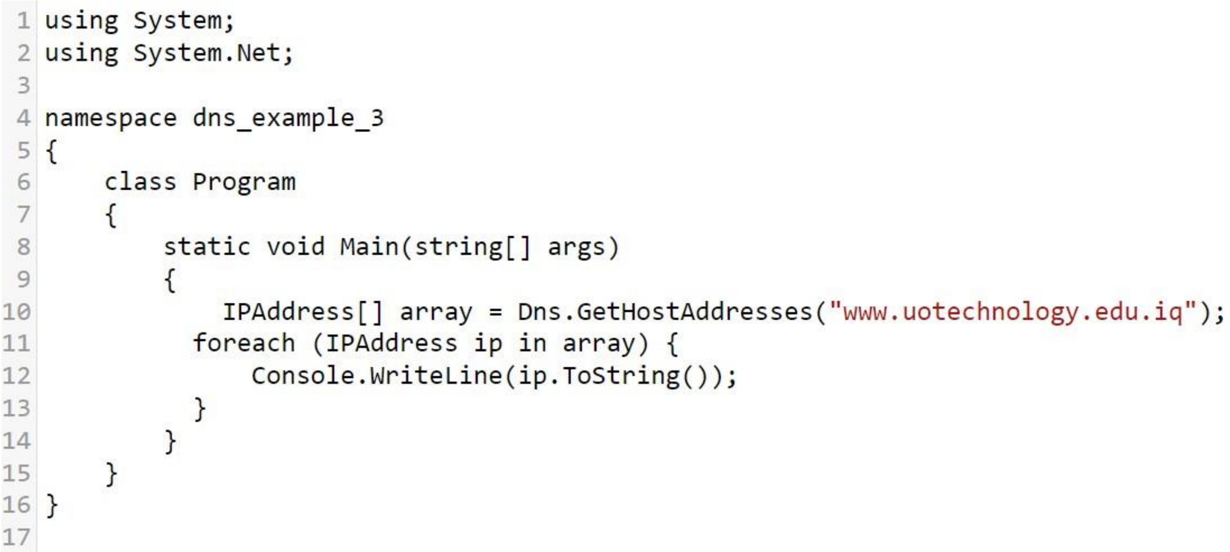
This C# program shows the Dns class and the Dns.GetHostAddresses method, DNS servers resolve host names to IP addresses.

With the System.Net namespace in the .NET Framework, we easily perform this task. The Dns.GetHostAddresses method converts a host name to its available IP addresses.

Namespace: System.Net

Example

To get started, add the System.Netname space to the top of your program. Next, we use the host name "www.uotechnology.edu.iq" and pass that string literal to the Dns.GetHostAddresses method. This returns an array of IP Address classes.

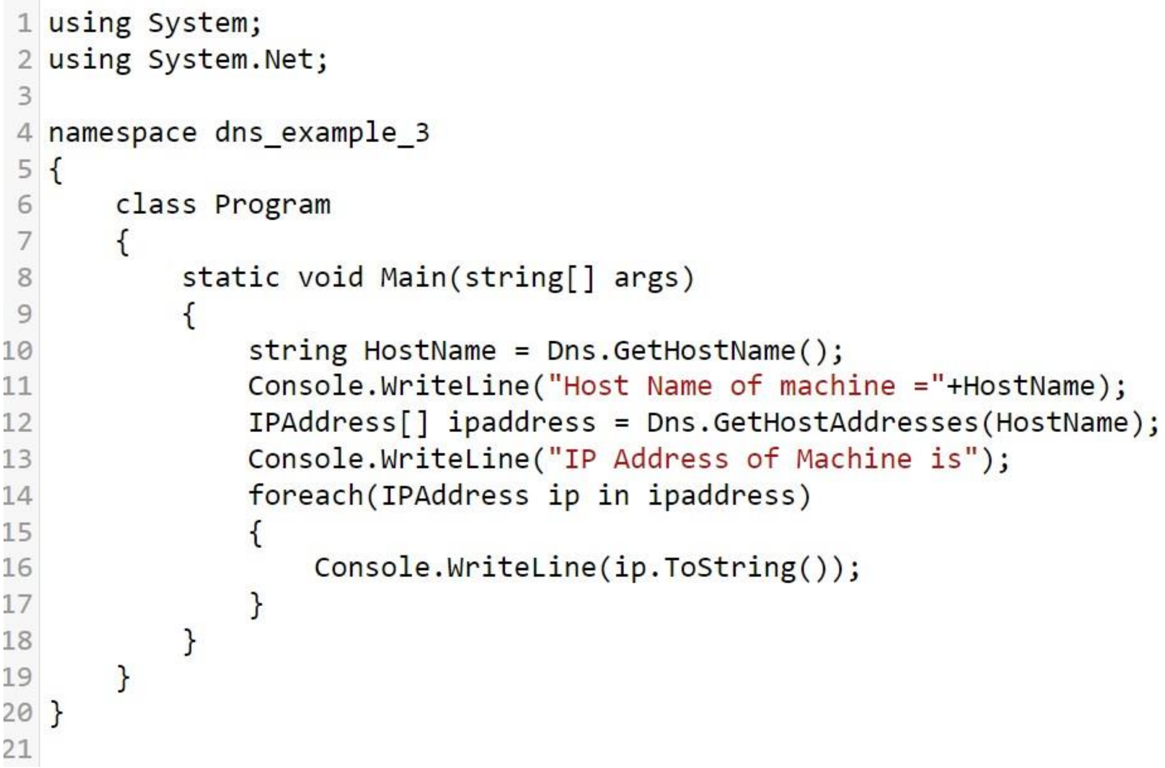


```
using System;
using System.Net;

namespace dns_example_3
{
    class Program
    {
        static void Main(string[] args)
        {
            IPAddress[] array = Dns.GetHostAddresses("www.uotechnology.edu.iq");
          foreach (IPAddress ip in array) {
              Console.WriteLine(ip.ToString());
          }
        }
    }
}
```

Example:

Program for getting the IP Address of Host(Local).

The Dns class provides the GetHostAddresses() method that takes HostName as a parameter and returns an array of IPAddress. We get both IPv4 and IPv6 of the machine.



```csharp
using System;
using System.Net;

namespace dns_example_3
{
    class Program
    {
        static void Main(string[] args)
        {
            string HostName = Dns.GetHostName();
            Console.WriteLine("Host Name of machine ="+HostName);
            IPAddress[] ipaddress = Dns.GetHostAddresses(HostName);
            Console.WriteLine("IP Address of Machine is");
            foreach(IPAddress ip in ipaddress)
            {
                Console.WriteLine(ip.ToString());
            }
        }
    }
}

```

IPHostEntryClass

Provides a container class for Internet host address information, The IPHostEntry class associates a Domain Name System (DNS) host name with an array of aliases and an array of matching IP addresses; The IPHostEntry class is used as a helper class with the Dns class.

Namespace: System.Net

Constructors

IPHostEntry() Initializes a new instance of the IPHostEntry class.

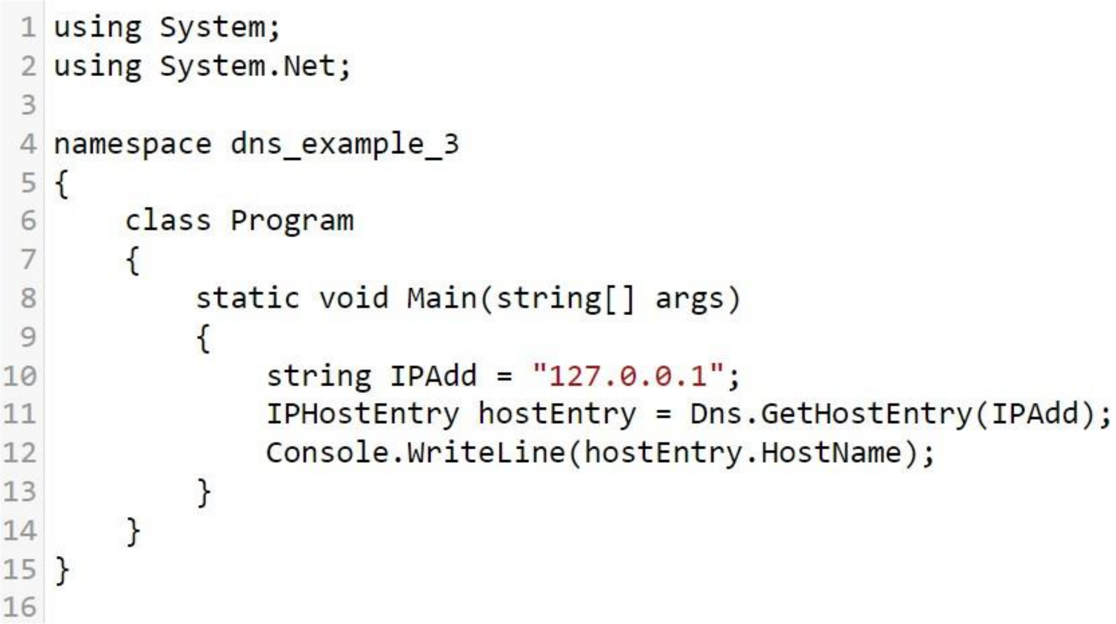
Properties

Address List	Gets or sets a list of IP addresses that are associated with a host.
Aliases	Gets or sets a list of aliases that are associated with a host.
Host Name	Gets or sets the DNS name of the host.

Example

Program for getting HostName based on IPAddress.

The GetHostEntry() method of the DNS class takes IPAddress as parameter and returnsIPHostEntrythatcontainsaddressinformationaboutthehostspecifiedinthe address.



```
using System;
using System.Net;

namespace dns_example_3
{
    class Program
    {
        static void Main(string[] args)
        {
            string IPAdd = "127.0.0.1";
            IPHostEntry hostEntry = Dns.GetHostEntry(IPAdd);
            Console.WriteLine(hostEntry.HostName);
        }
    }
}

```

IPHostEntry.Address List Property

Gets or sets a list of IP addresses that are associated with a host.

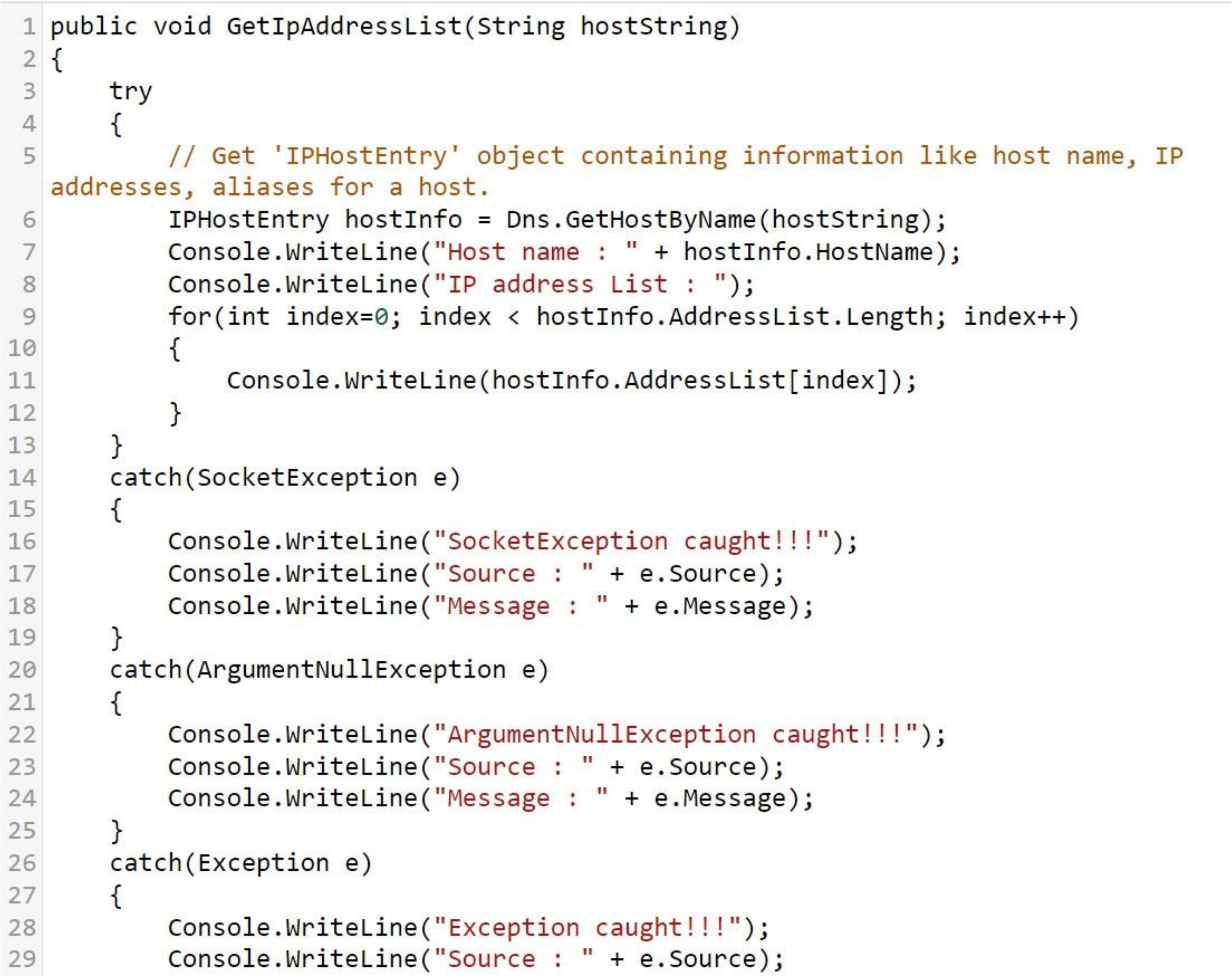
Property Value

IPAddress[]

An array of type IPAddress that contains IP addresses that resolve to the host names that are contained in the Aliases property.

Examples

The following example uses the Address List property to access the IP addresses that are associated with the IPHostEntry.



```
public void GetIpAddressList(String hostString)
{
    try
    {
        // Get 'IPHostEntry' object containing information like host name, IP addresses, aliases for a host.
        IPHostEntry hostInfo = Dns.GetHostByName(hostString);
        Console.WriteLine("Host name : " + hostInfo.HostName);
        Console.WriteLine("IP address List : ");
        for(int index=0; index < hostInfo.AddressList.Length; index++)
        {
            Console.WriteLine(hostInfo.AddressList[index]);
        }
    }
    catch(SocketException e)
    {
        Console.WriteLine("SocketException caught!!!");
        Console.WriteLine("Source : " + e.Source);
        Console.WriteLine("Message : " + e.Message);
    }
    catch(ArgumentNullException e)
    {
        Console.WriteLine("ArgumentNullException caught!!!");
        Console.WriteLine("Source : " + e.Source);
        Console.WriteLine("Message : " + e.Message);
    }
    catch(Exception e)
    {
        Console.WriteLine("Exception caught!!!");
        Console.WriteLine("Source : " + e.Source);
```

IP Address Class

Initializes a new instance of the IPAddressclass, The IPAddressis created with the Address property set to address; If the length of address is 4, IPAddress(Byte[]) constructs an IPv4 address; otherwise, an IPv6 address with a scope of 0 is constructed.

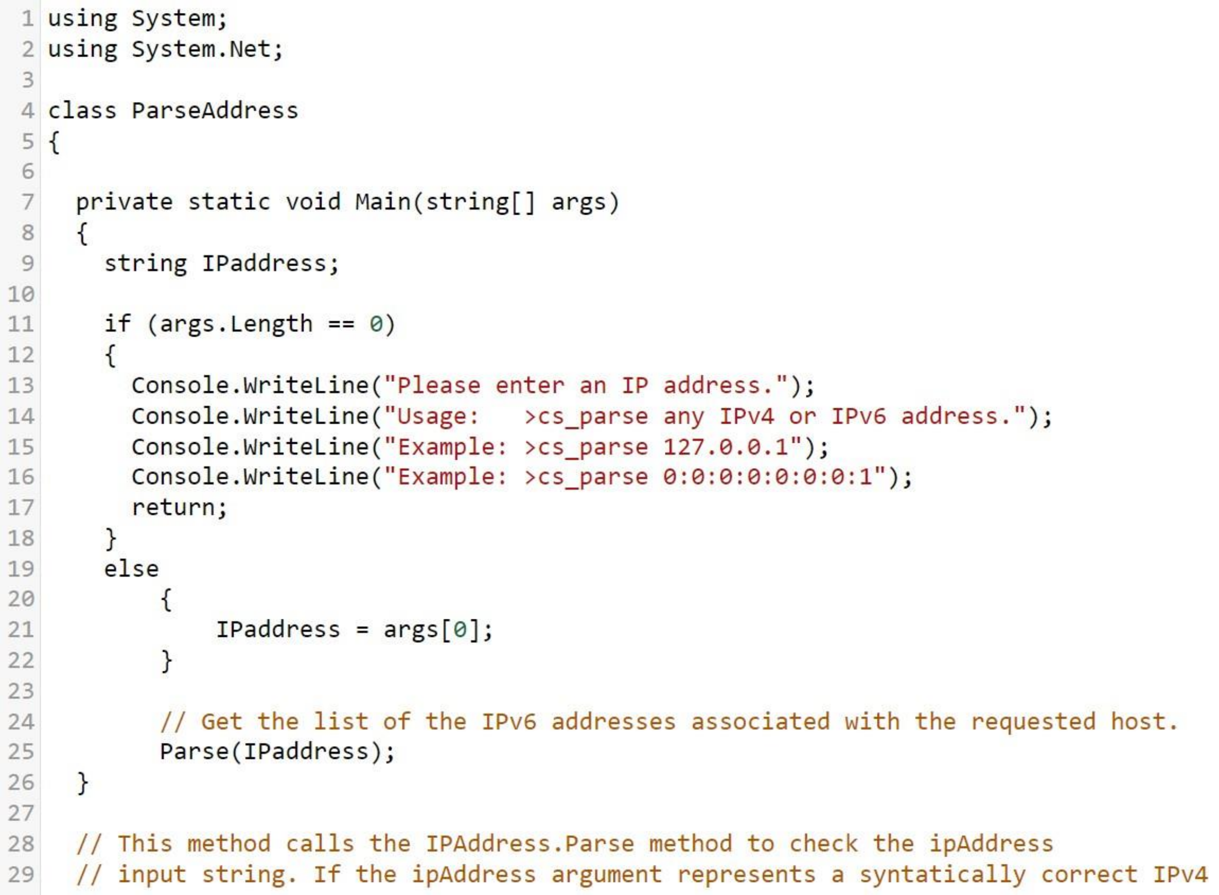
Namespace: System.Net

IPAddress.ParseMethod

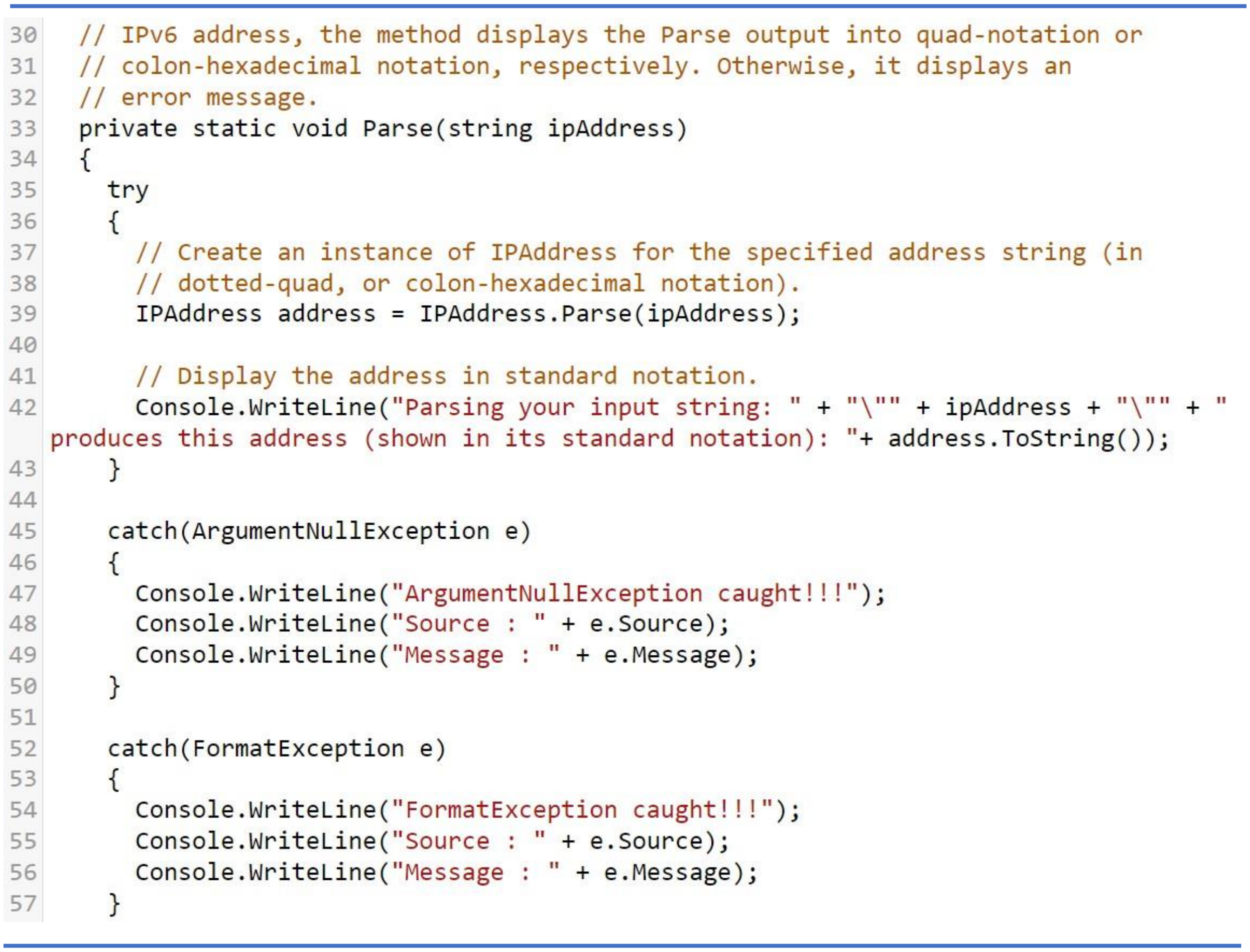
Converts an IP address string to an IP Address instance.

Examples

The following code converts a string that contains an IP address, in dotted-quad notation for IPv4 or in colon-hexadecimal notation forIPv6, into an instance of the IPAddressclass.Thenitusestheoverloaded ToStringmethodtodisplaytheaddress in standard notation.



```
using System;
using System.Net;

class ParseAddress
{

  private static void Main(string[] args)
  {
    string IPaddress;

    if (args.Length == 0)
    {
      Console.WriteLine("Please enter an IP address.");
      Console.WriteLine("Usage:    >cs_parse any IPv4 or IPv6 address.");
      Console.WriteLine("Example: >cs_parse 127.0.0.1");
      Console.WriteLine("Example: >cs_parse 0:0:0:0:0:0:0:1");
      return;
    }
    else
        {
            IPaddress = args[0];
        }

        // Get the list of the IPv6 addresses associated with the requested host.
        Parse(IPaddress);
  }

  // This method calls the IPAddress.Parse method to check the ipAddress
  // input string. If the ipAddress argument represents a syntatically correct IPv4
```



```
  // IPv6 address, the method displays the Parse output into quad-notation or
  // colon-hexadecimal notation, respectively. Otherwise, it displays an
  // error message.
  private static void Parse(string ipAddress)
  {
    try
    {
      // Create an instance of IPAddress for the specified address string (in
      // dotted-quad, or colon-hexadecimal notation).
      IPAddress address = IPAddress.Parse(ipAddress);

      // Display the address in standard notation.
      Console.WriteLine("Parsing your input string: " + "\"" + ipAddress + "\"" + " produces this address (shown in its standard notation): "+ address.ToString());
    }

    catch(ArgumentNullException e)
    {
      Console.WriteLine("ArgumentNullException caught!!!");
      Console.WriteLine("Source : " + e.Source);
      Console.WriteLine("Message : " + e.Message);
    }

    catch(FormatException e)
    {
      Console.WriteLine("FormatException caught!!!");
      Console.WriteLine("Source : " + e.Source);
      Console.WriteLine("Message : " + e.Message);
    }
```

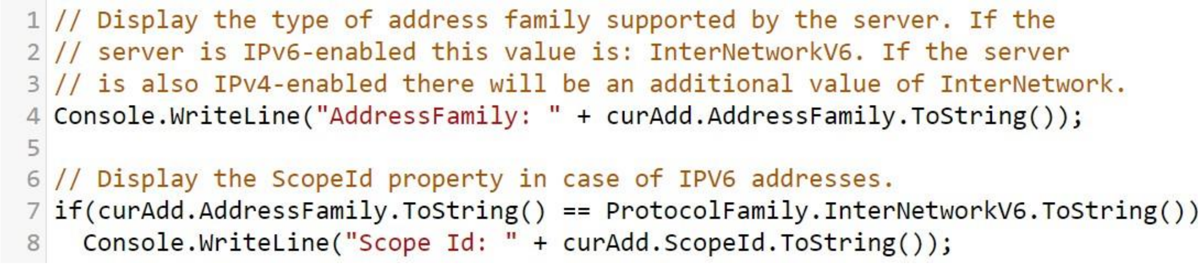
IPAddress.AddressFamilyProperty

Gets the address family of the IP address.

Property Value

Address Family Returns InterNetwork for IPv4 or InterNetwork V6 for IPv6.

Examples Refer to the example in the IPAddress class topic.



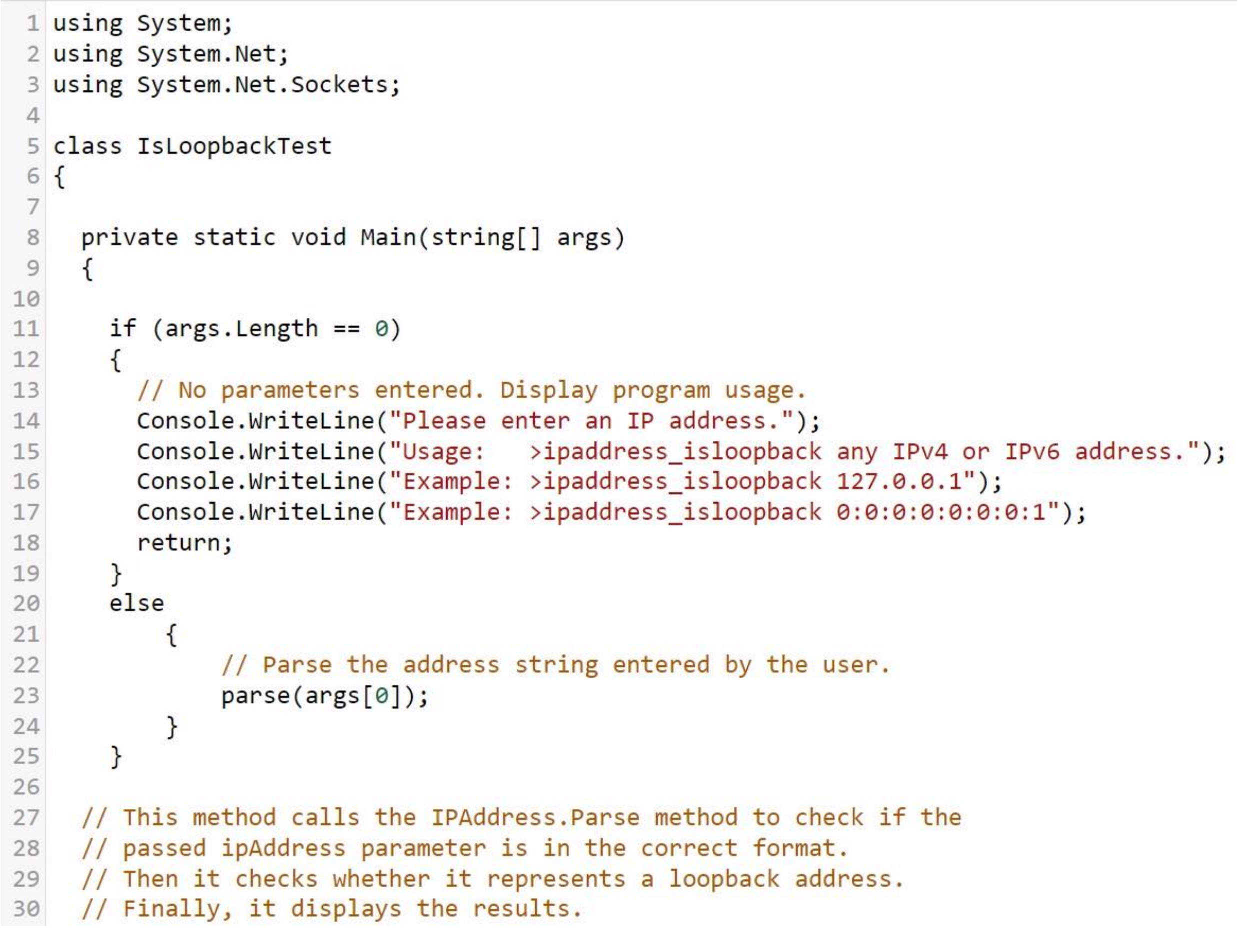
```
// Display the type of address family supported by the server. If the
// server is IPv6-enabled this value is: InterNetworkV6. If the server
// is also IPv4-enabled there will be an additional value of InterNetwork.
Console.WriteLine("AddressFamily: " + curAdd.AddressFamily.ToString());

// Display the ScopeId property in case of IPV6 addresses.
if(curAdd.AddressFamily.ToString() == ProtocolFamily.InterNetworkV6.ToString())
  Console.WriteLine("Scope Id: " + curAdd.ScopeId.ToString());
```

IPAddress.IsLoopback(IPAddress)Method

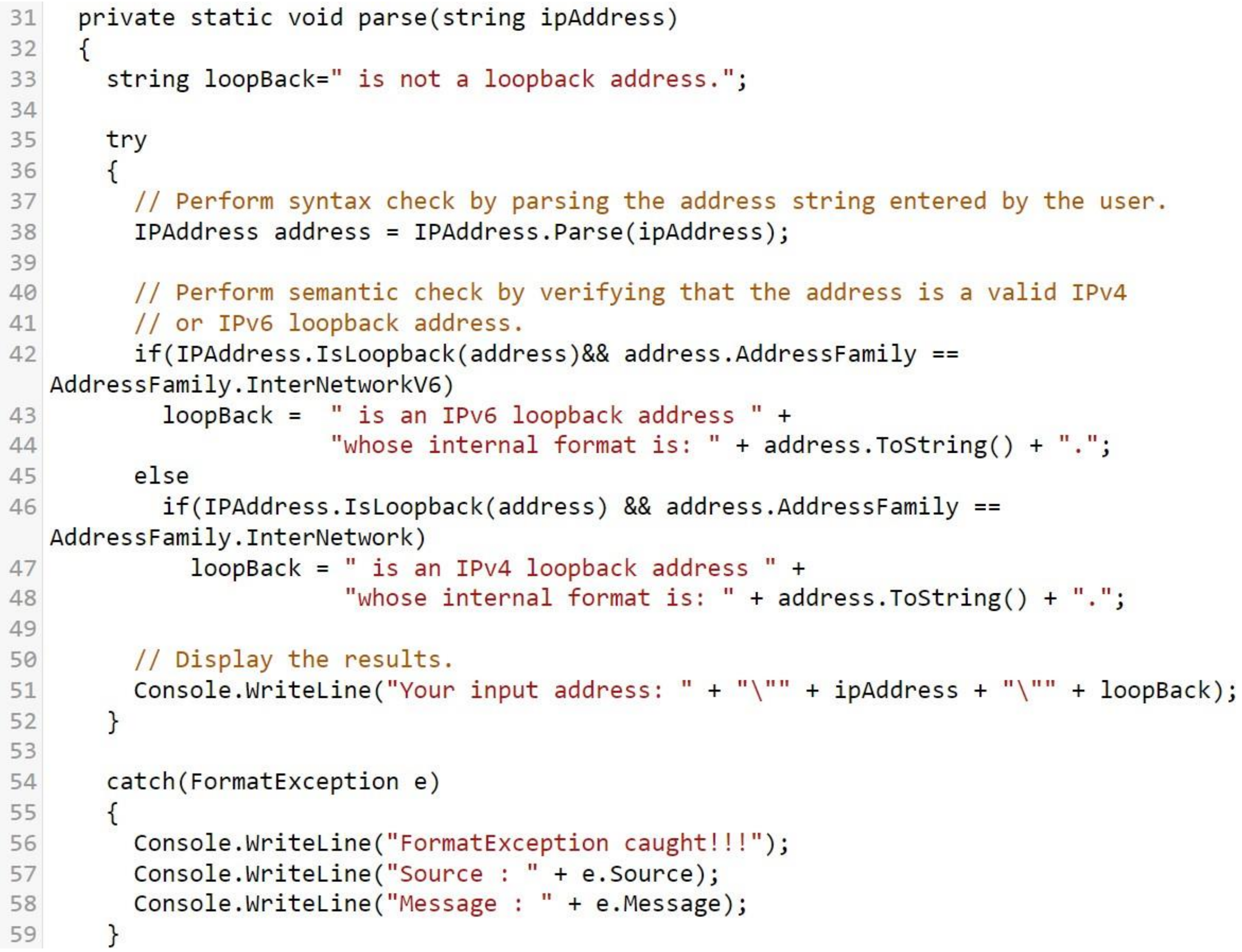
IndicateswhetherthespecifiedIPaddressistheloopback address.

Examples

Thefollowing codeexampleusestheIsLoopback methodtodeterminewhetherthe specified address is a loopback address.



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

class IsLoopbackTest
{

  private static void Main(string[] args)
  {

    if (args.Length == 0)
    {
      // No parameters entered. Display program usage.
      Console.WriteLine("Please enter an IP address.");
      Console.WriteLine("Usage:   >ipaddress_isloopback any IPv4 or IPv6 address.");
      Console.WriteLine("Example: >ipaddress_isloopback 127.0.0.1");
      Console.WriteLine("Example: >ipaddress_isloopback 0:0:0:0:0:0:0:1");
      return;
    }
    else
        {
            // Parse the address string entered by the user.
            parse(args[0]);
        }
    }

  // This method calls the IPAddress.Parse method to check if the
  // passed ipAddress parameter is in the correct format.
  // Then it checks whether it represents a loopback address.
  // Finally, it displays the results.
```



```
private static void parse(string ipAddress)
{
  string loopBack=" is not a loopback address.";

  try
  {
    // Perform syntax check by parsing the address string entered by the user.
    IPAddress address = IPAddress.Parse(ipAddress);

    // Perform semantic check by verifying that the address is a valid IPv4
    // or IPv6 loopback address.
    if(IPAddress.IsLoopback(address)&& address.AddressFamily ==
AddressFamily.InterNetworkV6)
      loopBack =  " is an IPv6 loopback address " +
                  "whose internal format is: " + address.ToString() + ".";
    else
      if(IPAddress.IsLoopback(address) && address.AddressFamily ==
AddressFamily.InterNetwork)
        loopBack = " is an IPv4 loopback address " +
                   "whose internal format is: " + address.ToString() + ".";

    // Display the results.
    Console.WriteLine("Your input address: " + "\"" + ipAddress + "\"" + loopBack);
  }

  catch(FormatException e)
  {
    Console.WriteLine("FormatException caught!!!");
    Console.WriteLine("Source : " + e.Source);
    Console.WriteLine("Message : " + e.Message);
  }
```

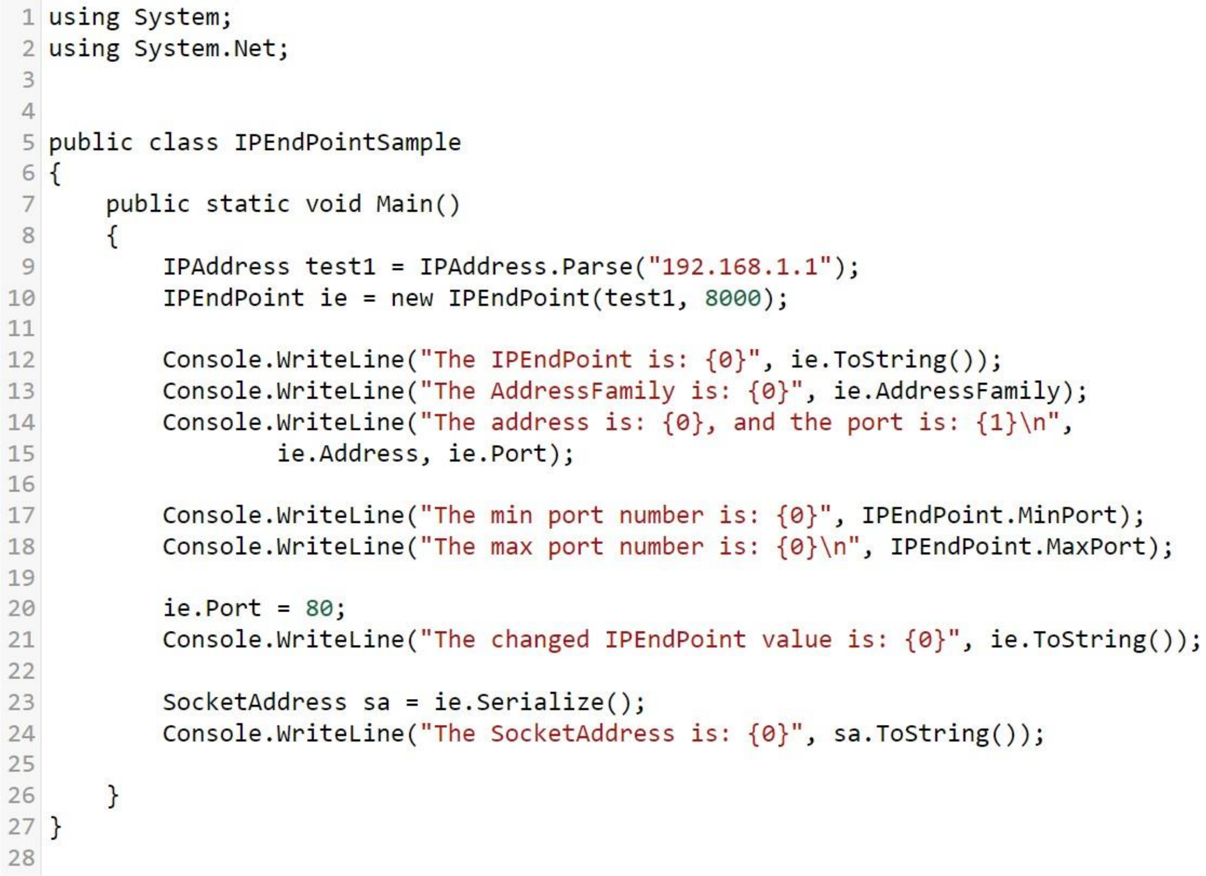
IPEndPointClass

The IPEndPoint class contains the host and local or remote port information neededbyanapplicationtoconnecttoaserviceonahost.Bycombiningthehost's IP address and port number of a service, the IPEndPoint class forms a connection point to a service.

Namespace: System.Net

Properties

Address	GetsorsetstheIPaddressofthe endpoint.
AddressFamily	GetstheInternetProtocol(IP)addressfamily.
Port	Getsorsetstheportnumberofthe endpoint.



```
using System;
using System.Net;


public class IPEndPointSample
{
    public static void Main()
    {
        IPAddress test1 = IPAddress.Parse("192.168.1.1");
        IPEndPoint ie = new IPEndPoint(test1, 8000);

        Console.WriteLine("The IPEndPoint is: {0}", ie.ToString());
        Console.WriteLine("The AddressFamily is: {0}", ie.AddressFamily);
        Console.WriteLine("The address is: {0}, and the port is: {1}\n",
                ie.Address, ie.Port);

        Console.WriteLine("The min port number is: {0}", IPEndPoint.MinPort);
        Console.WriteLine("The max port number is: {0}\n", IPEndPoint.MaxPort);

        ie.Port = 80;
        Console.WriteLine("The changed IPEndPoint value is: {0}", ie.ToString());

        SocketAddress sa = ie.Serialize();
        Console.WriteLine("The SocketAddress is: {0}", sa.ToString());

    }
}

```

SocketClass

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

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

If your application only requires one thread during execution, use the following methods, which are designed for synchronous operation mode.

- Ifyouareusingaconnection-orientedprotocolsuchasTCP,yourservercan listenforconnectionsusingtheListenmethod.TheAcceptmethod processes any incoming connection requests and returns a Socket that you canusetocommunicatedatawiththeremotehost.UsethisreturnedSocket to call the Send or Receive method. Call the Bind method prior to calling the Listen method if you want to specify the local IP address and port number. Use a port number of zero if you want the underlying service provider to assign a free port for you. If you want to connect to a listening host, call the Connect method. To communicate data, call the Send or Receive method.

- If you are using a connectionless protocol such as UDP, you do not need to listen for connections at all. Call the ReceiveFrom method to accept any incomingdatagrams.UsetheSendTomethodtosenddatagramstoaremote host.

To process communications using separate threads during execution, use the following methods, which are designed for asynchronous operation mode.

- If you are using a connection-oriented protocol such as TCP, use the Socket, BeginConnect,andEndConnectmethodstoconnectwithalisteninghost.Use the BeginSend and EndSend or BeginReceive and EndReceive methods to communicate data asynchronously. Incoming connection requests can be processed using BeginAccept and EndAccept.
- If you are using a connectionless protocol such as UDP, you can use BeginSendToandEndSendTotosenddatagrams,andBeginReceiveFromand EndReceiveFrom to receive datagrams.

Namespace: System.Net.Sockets

Methods

Accept()	CreatesanewSocketforanewlycreated connection.
Bind(EndPoint)	AssociatesaSocketwithalocal endpoint.
Close()	ClosestheSocketconnectionandreleasesall associated resources.
Connect(EndPoint)	Establishesaconnectiontoaremotehost.

Connect(IPAddress,Int32)	Establishes a connection to a remote host. ThehostisspecifiedbyanIPaddressanda port number.
Dispose()	Releasesallresourcesusedbythecurrent instance of the Socket class.
Listen()	PlacesaSocketinalisteningstate.
Receive(Byte[])	ReceivesdatafromaboundSocketintoa receive buffer.
Send(Byte[])	Sendsdatatoaconnected Socket.
Send(Byte[],Int32,SocketFlags)	Sendsthespecifiednumberofbytesofdata to a connected Socket, using the specified SocketFlags.
Shutdown(SocketShutdown)	DisablessendsandreceivesonaSocket.
ToString()	Returnsastringthatrepresentsthecurrent object.

Socket.SendMethod

Sendsdatatoaconnected Socket.

Send(Byte[],Int32,Int32,SocketFlags,SocketError)

Sendsthespecifiednumberofbytesofdatatoaconnected Socket,startingatthe specified offset, and using the specified SocketFlags.

Parameters

Buffer Byte[]

AnarrayoftypeBytethatcontainsthedatatobe sent.

Offset Int32
Thepositioninthedatabufferatwhichtobeginsending data.

Size Int32
Thenumberofbytestosend.

socketFlags SocketFlags
AbitwisecombinationoftheSocketFlagsvalues.

errorCode SocketError

ASocketErrorobjectthatstoresthesocketerror.

SendsynchronouslysendsdatatotheremotehostspecifiedintheConnectorAccept methodandreturnsthenumberofbytessuccessfullysent.Sendcanbeusedforboth connection-oriented and connectionless protocols.

Socket.ReceiveMethod

Receivesdatafromabound Socket.

Receive(Byte[],Int32,Int32,SocketFlags,SocketError)

ReceivesdatafromaboundSocketintoareceivebuffer,usingthespecified SocketFlags.

Parameters

Buffer Byte[]

AnarrayoftypeBytethatisthestoragelocationforthereceiveddata.

Offset Int32

Thepositioninthebufferparametertostorethereceiveddata.

Size Int32

Thenumberofbytestoreceive.

socketFlags SocketFlags

AbitwisecombinationoftheSocketFlagsvalues.

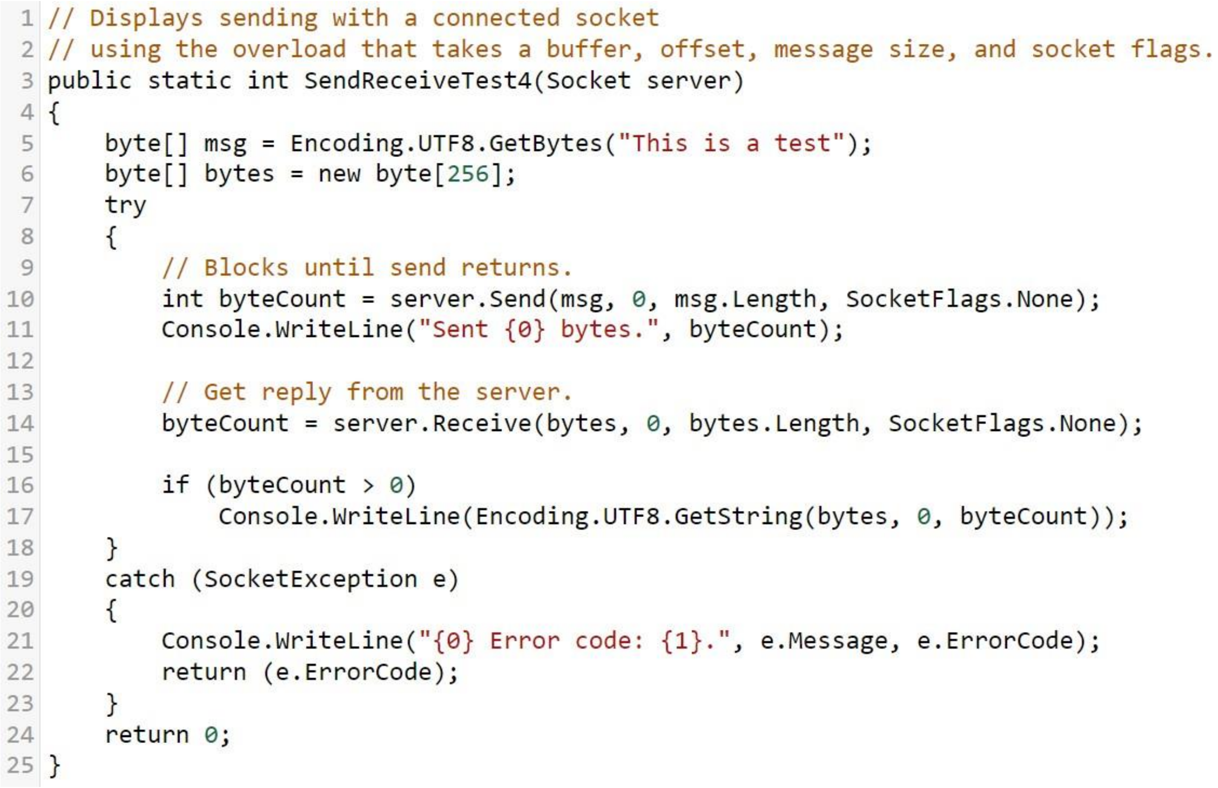
errorCode SocketError

ASocketErrorobjectthatstoresthesocketerror.

The Receive methodreadsdata intothebuffer parameterand returnsthe numberof bytes successfully read. You can call Receive from both connection-oriented and connectionless sockets.

Examples

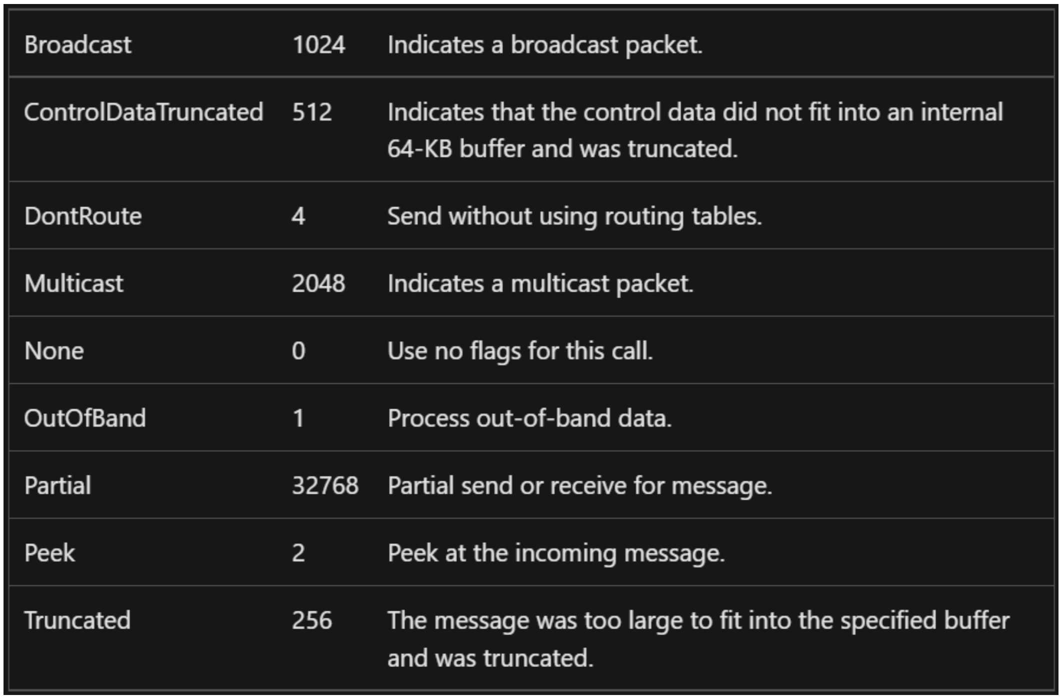
Thefollowingcodeexamplesspecifiesthedatabuffer,anoffset,asize,and SocketFlags for sending and receiving data to a connected Socket.



```
// Displays sending with a connected socket
// using the overload that takes a buffer, offset, message size, and socket flags.
public static int SendReceiveTest4(Socket server)
{
    byte[] msg = Encoding.UTF8.GetBytes("This is a test");
    byte[] bytes = new byte[256];
    try
    {
        // Blocks until send returns.
        int byteCount = server.Send(msg, 0, msg.Length, SocketFlags.None);
        Console.WriteLine("Sent {0} bytes.", byteCount);

        // Get reply from the server.
        byteCount = server.Receive(bytes, 0, bytes.Length, SocketFlags.None);

        if (byteCount > 0)
            Console.WriteLine(Encoding.UTF8.GetString(bytes, 0, byteCount));
    }
    catch (SocketException e)
    {
        Console.WriteLine("{0} Error code: {1}.", e.Message, e.ErrorCode);
        return (e.ErrorCode);
    }
    return 0;
}
```

SocketFlagsEnum

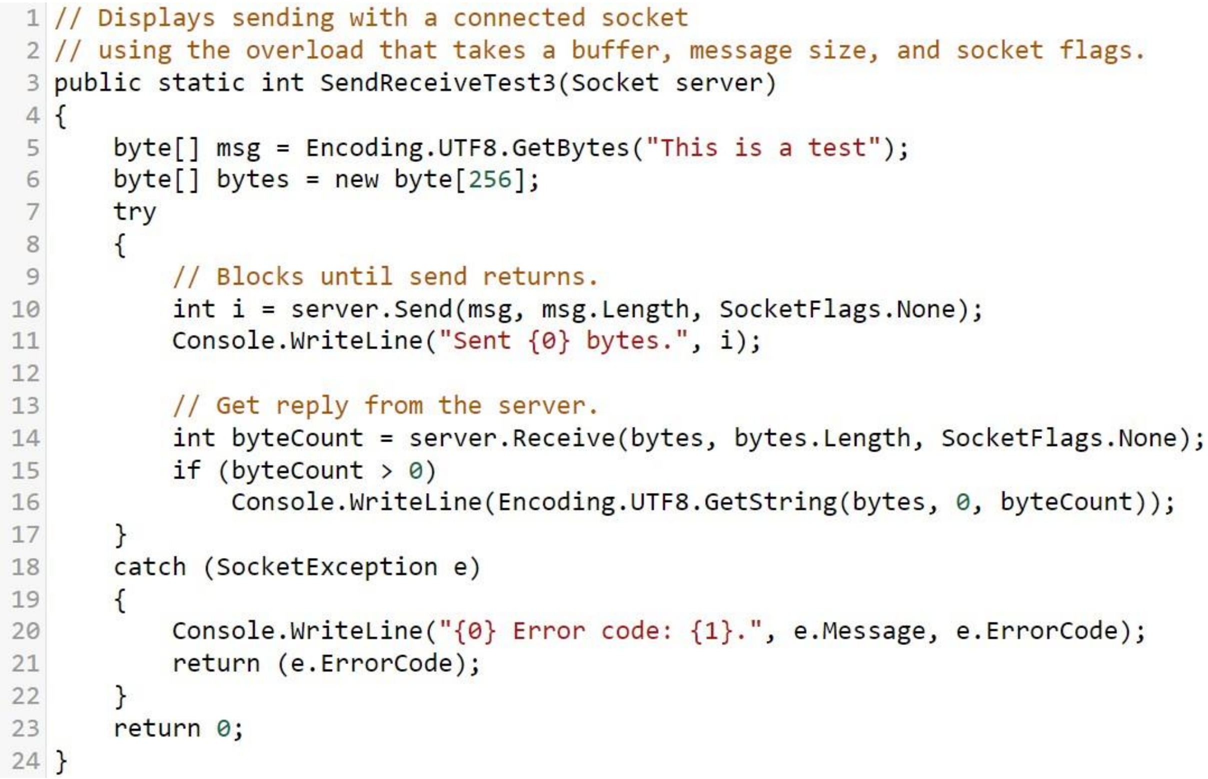
Thisenumerationsupportsabitwisecombinationofitsmembervalues,specifies socket send and receive behaviors.



Broadcast	1024	Indicates a broadcast packet.
ControlDataTruncated	512	Indicates that the control data did not fit into an internal 64-KB buffer and was truncated.
DontRoute	4	Send without using routing tables.
Multicast	2048	Indicates a multicast packet.
None	0	Use no flags for this call.
OutOfBand	1	Process out-of-band data.
Partial	32768	Partial send or receive for message.
Peek	2	Peek at the incoming message.
Truncated	256	The message was too large to fit into the specified buffer and was truncated.

Examples

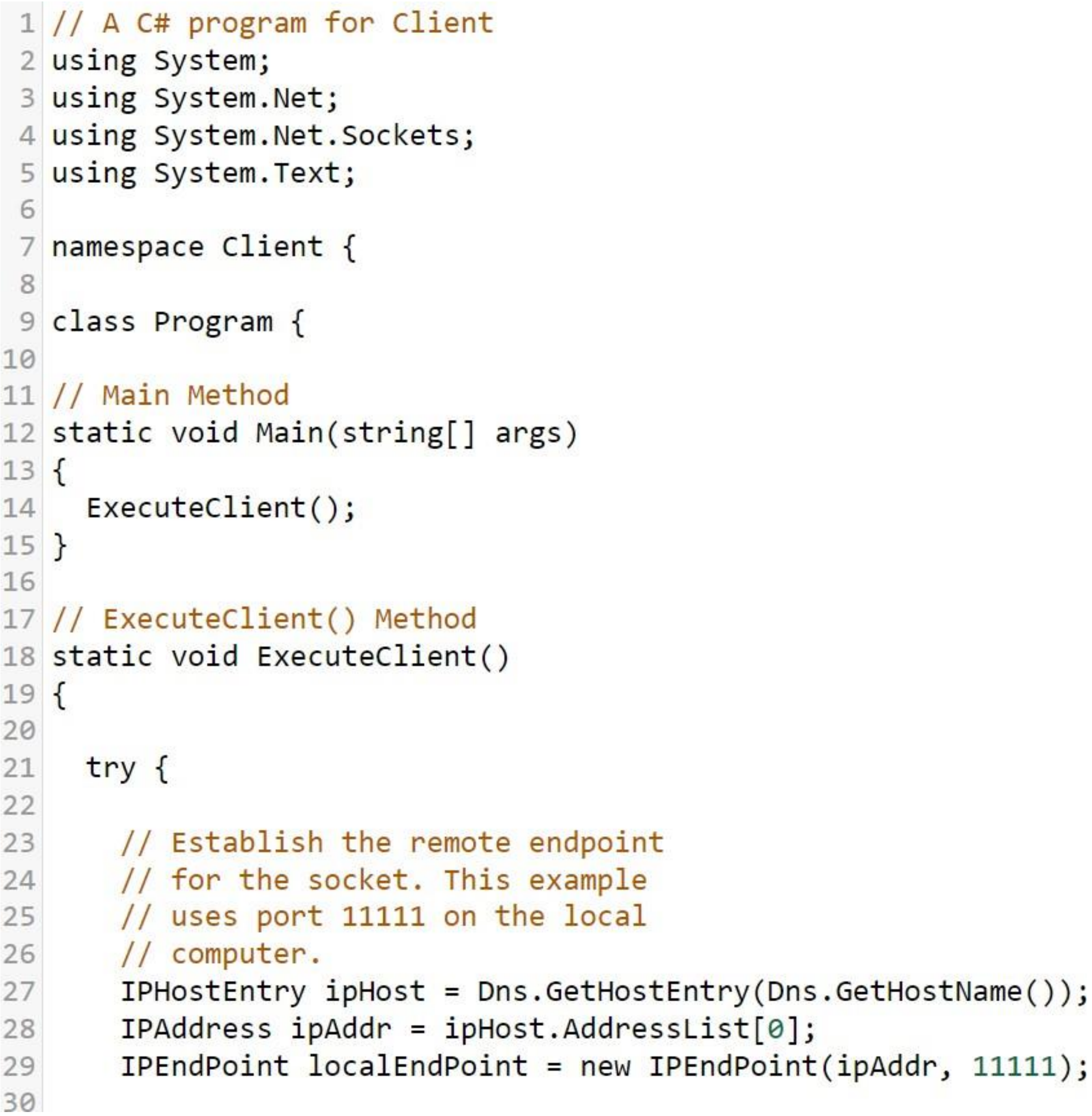
Thefollowingexamplesendsdataandspecifies SocketFlags.None.



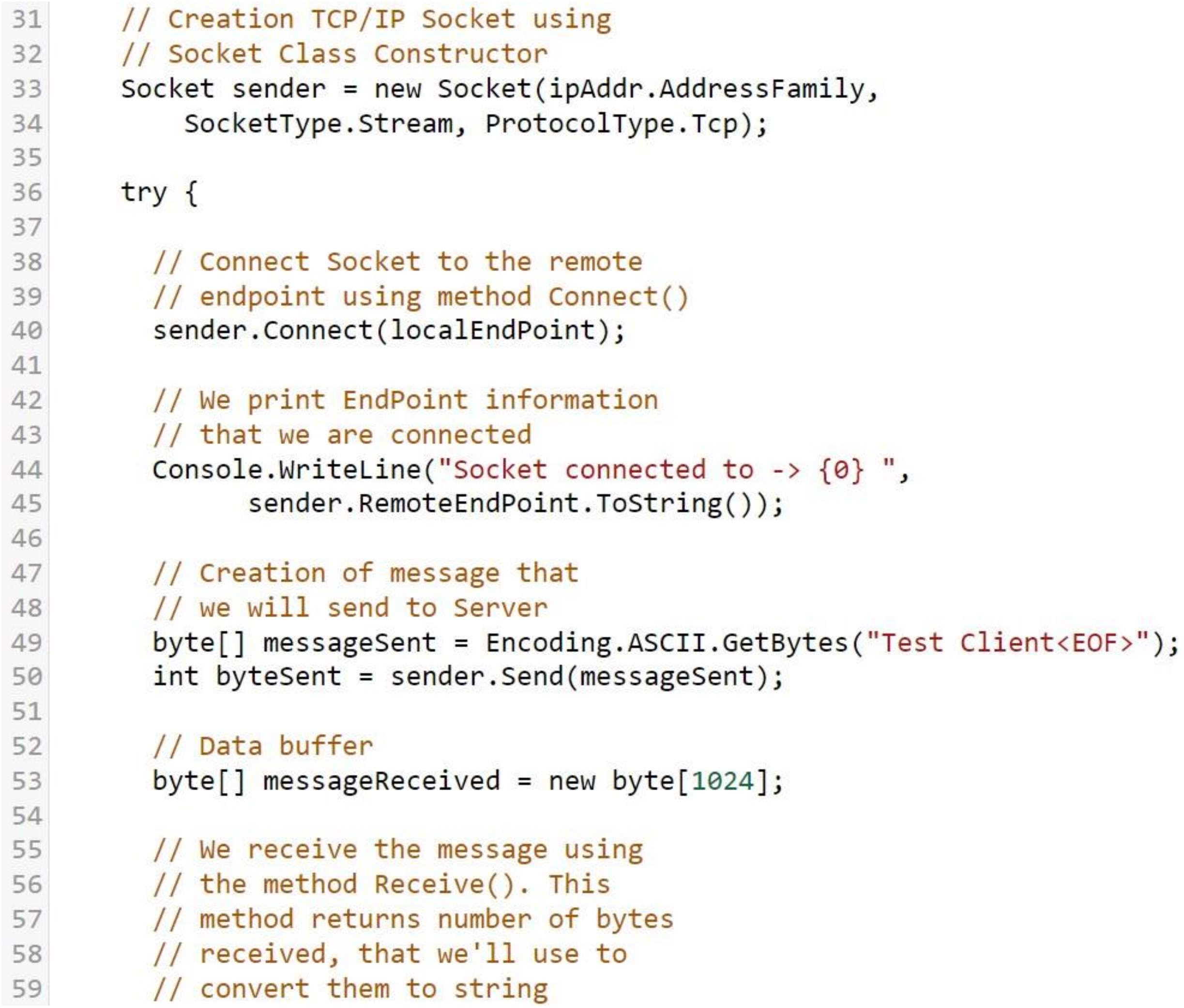
```
// Displays sending with a connected socket
// using the overload that takes a buffer, message size, and socket flags.
public static int SendReceiveTest3(Socket server)
{
    byte[] msg = Encoding.UTF8.GetBytes("This is a test");
    byte[] bytes = new byte[256];
    try
    {
        // Blocks until send returns.
        int i = server.Send(msg, msg.Length, SocketFlags.None);
        Console.WriteLine("Sent {0} bytes.", i);

        // Get reply from the server.
        int byteCount = server.Receive(bytes, bytes.Length, SocketFlags.None);
        if (byteCount > 0)
            Console.WriteLine(Encoding.UTF8.GetString(bytes, 0, byteCount));
    }
    catch (SocketException e)
    {
        Console.WriteLine("{0} Error code: {1}.", e.Message, e.ErrorCode);
        return (e.ErrorCode);
    }
    return 0;
}
```

ClientSideProgramming

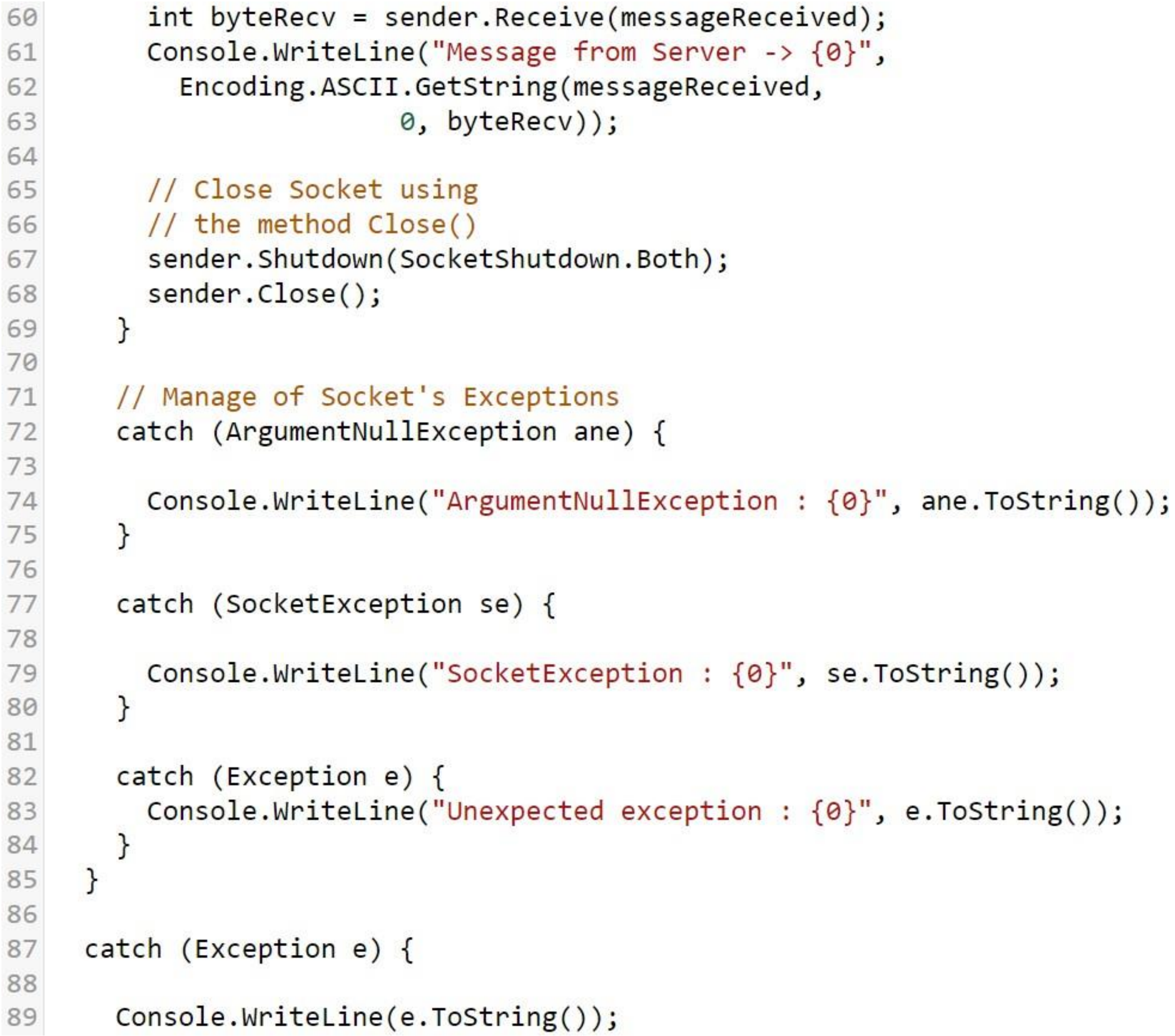
Before creating client’s socket, a user must decide what ‘IP Address‘ that he want to connect to, in this case, it is the localhost. At the same time, we also need the ‘Family‘ method that will belong to the socket itself. Then, through the ‘connect‘ method, we will connect the socket to the server. Before sending any message, it mustbe converted into a byte array.Then andonly then,it can be senttothe server throughthe‘send‘method.Later,thankstothe‘receive‘methodwearegoingtoget a byte array as answer by the server.



```
// A C# program for Client
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace Client {

class Program {

// Main Method
static void Main(string[] args)
{
  ExecuteClient();
}

// ExecuteClient() Method
static void ExecuteClient()
{

  try {

    // Establish the remote endpoint
    // for the socket. This example
    // uses port 11111 on the local
    // computer.
    IPHostEntry ipHost = Dns.GetHostEntry(Dns.GetHostName());
    IPAddress ipAddr = ipHost.AddressList[0];
    IPEndPoint localEndPoint = new IPEndPoint(ipAddr, 11111);

```



```
    // Creation TCP/IP Socket using
    // Socket Class Constructor
    Socket sender = new Socket(ipAddr.AddressFamily,
        SocketType.Stream, ProtocolType.Tcp);

    try {

      // Connect Socket to the remote
      // endpoint using method Connect()
      sender.Connect(localEndPoint);

      // We print EndPoint information
      // that we are connected
      Console.WriteLine("Socket connected to -> {0} ",
            sender.RemoteEndPoint.ToString());

      // Creation of message that
      // we will send to Server
      byte[] messageSent = Encoding.ASCII.GetBytes("Test Client<EOF>");
      int byteSent = sender.Send(messageSent);

      // Data buffer
      byte[] messageReceived = new byte[1024];

      // We receive the message using
      // the method Receive(). This
      // method returns number of bytes
      // received, that we'll use to
      // convert them to string
```

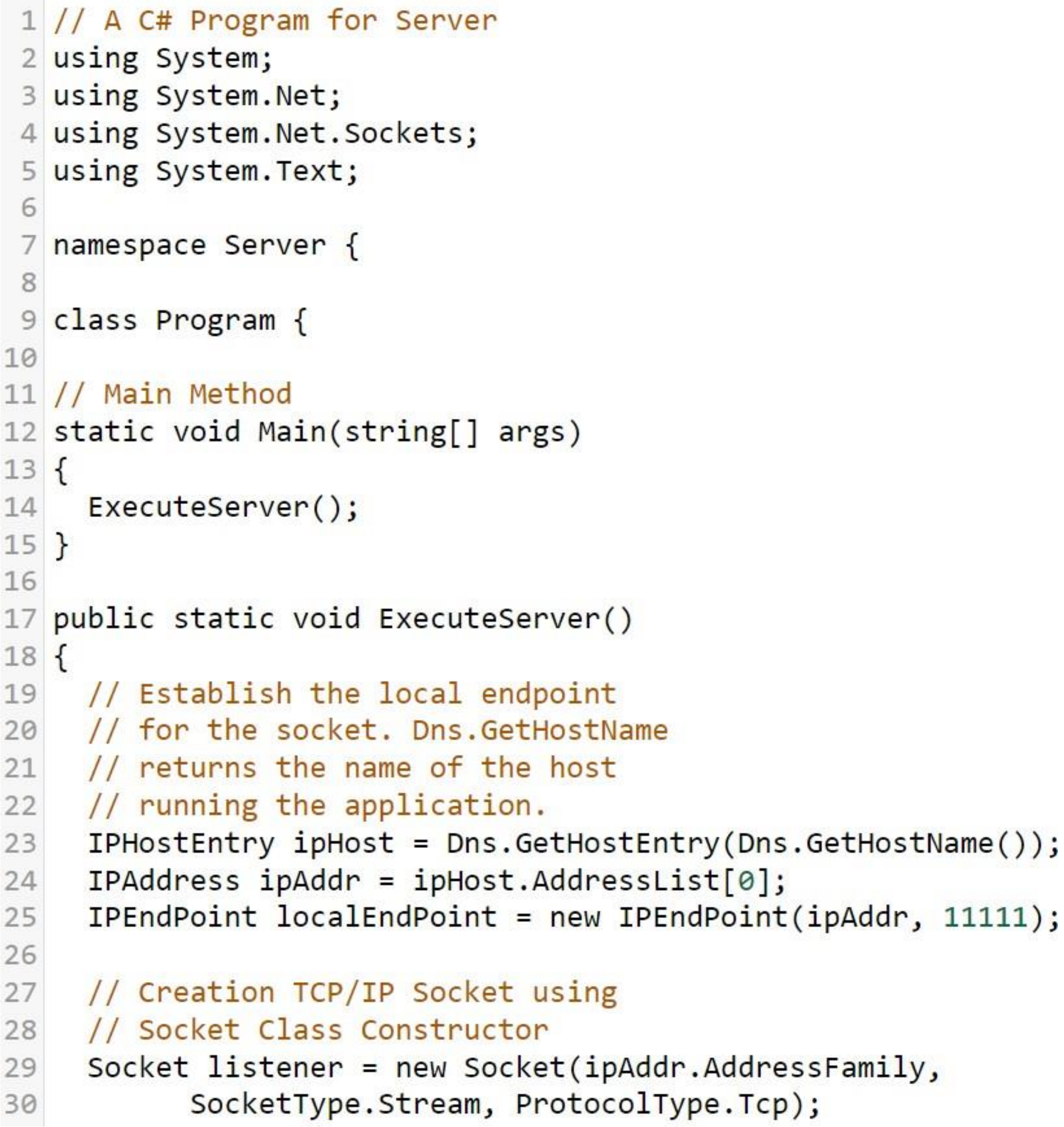


```
    int byteRecv = sender.Receive(messageReceived);
    Console.WriteLine("Message from Server -> {0}",
      Encoding.ASCII.GetString(messageReceived,
                    0, byteRecv));

    // Close Socket using
    // the method Close()
    sender.Shutdown(SocketShutdown.Both);
    sender.Close();
  }

  // Manage of Socket's Exceptions
  catch (ArgumentNullException ane) {

    Console.WriteLine("ArgumentNullException : {0}", ane.ToString());
  }

  catch (SocketException se) {

    Console.WriteLine("SocketException : {0}", se.ToString());
  }

  catch (Exception e) {
    Console.WriteLine("Unexpected exception : {0}", e.ToString());
  }
}

catch (Exception e) {

  Console.WriteLine(e.ToString());
```

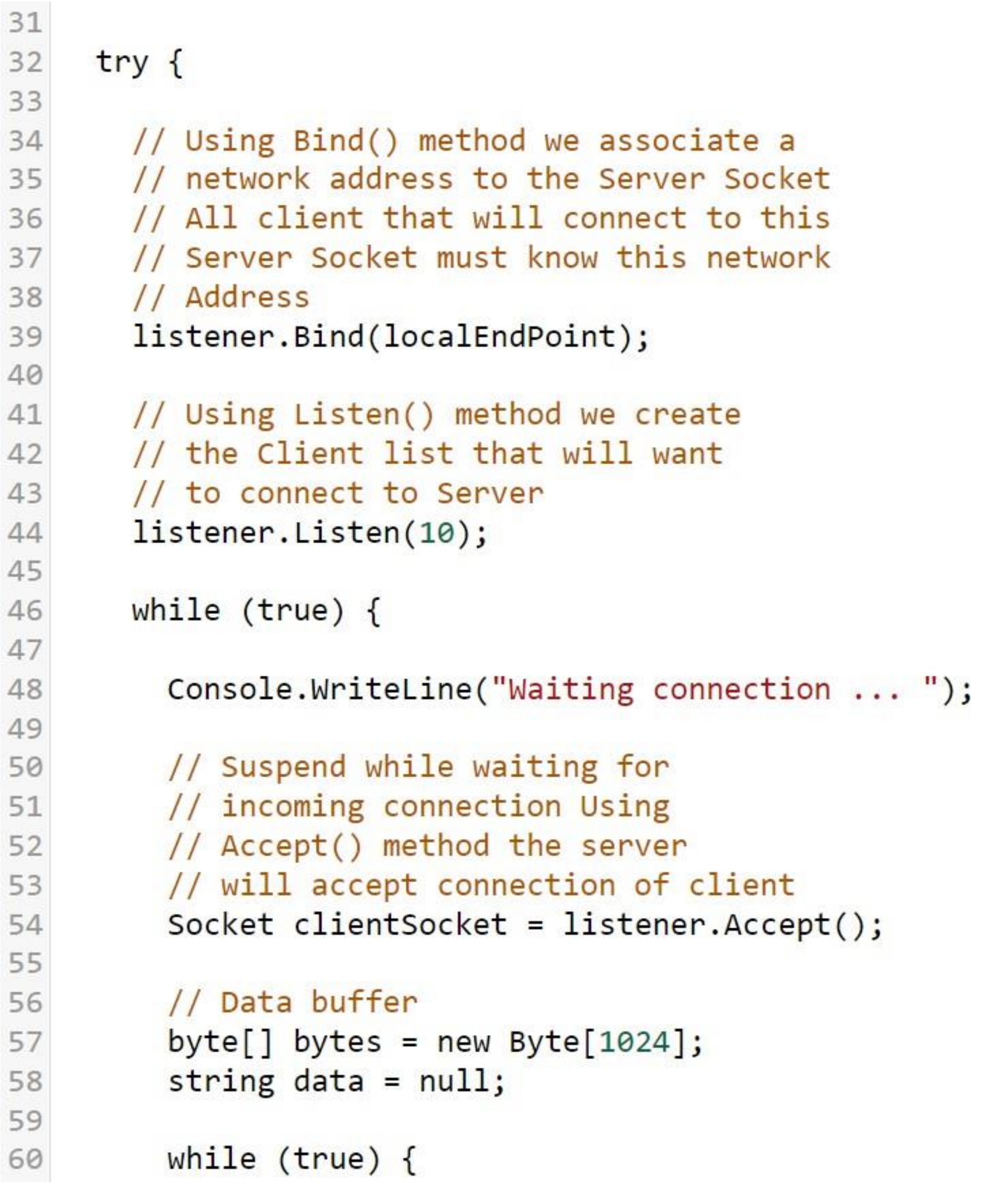
ServerSideProgramming

Inthesameway,weneedan‘IPaddress’thatidentifiestheserverinordertoletthe clients to connect.After creating the socket, we call the ‘bind‘ method which binds theIPtothesocket.Then,callthe‘listen‘method.Thisoperationisresponsiblefor creating the waiting queue which will be related to every opened ‘socket‘. The ‘listen‘ method takes as input the maximum number of clients that can stay in the waiting queue. As stated above, there is communication with the client through ‘send‘ and ‘receive‘ methods.

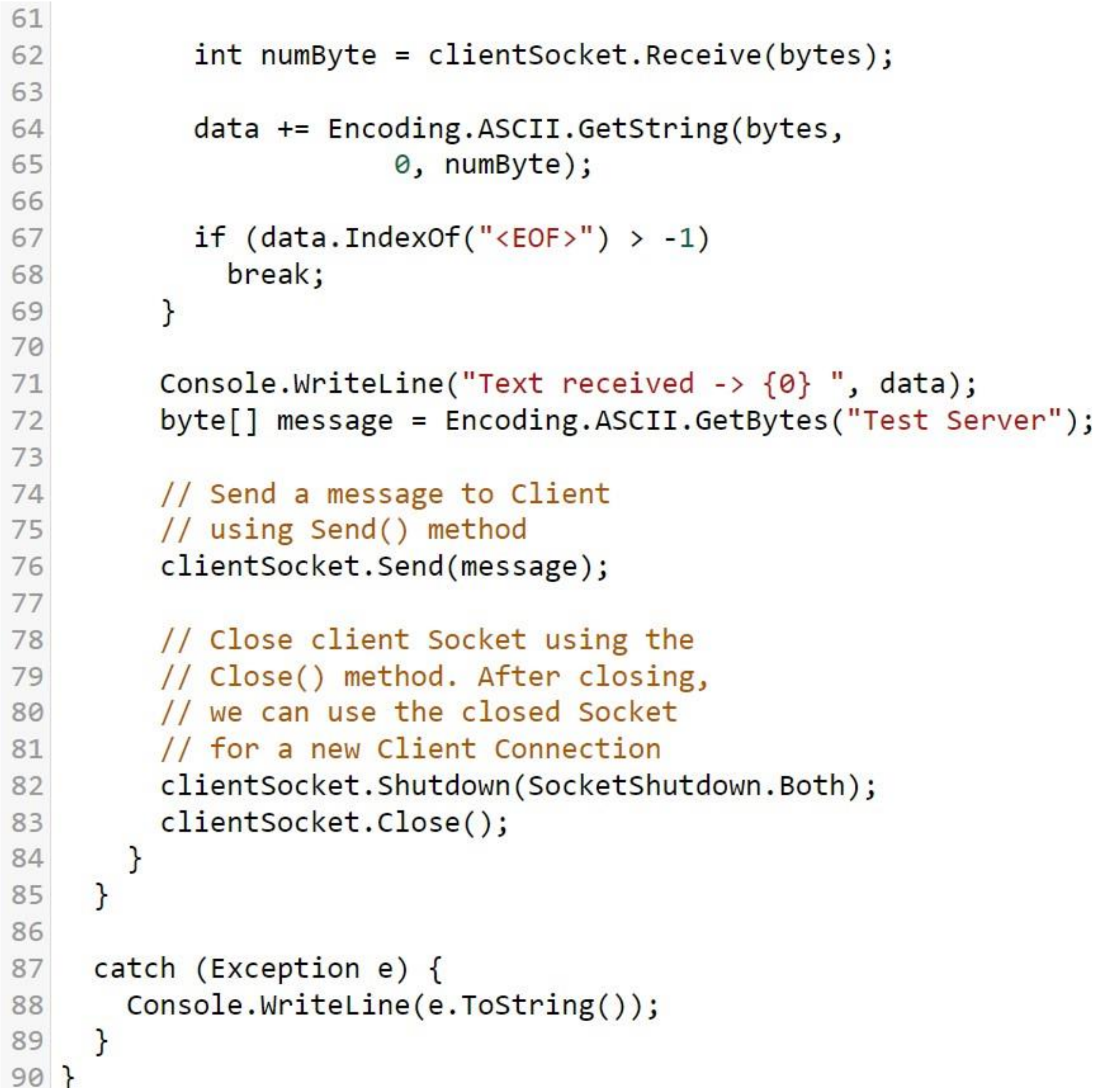
Note:Don’tforgettheconversionintoabytearray.



```
// A C# Program for Server
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace Server {

class Program {

// Main Method
static void Main(string[] args)
{
  ExecuteServer();
}

public static void ExecuteServer()
{
  // Establish the local endpoint
  // for the socket. Dns.GetHostName
  // returns the name of the host
  // running the application.
  IPHostEntry ipHost = Dns.GetHostEntry(Dns.GetHostName());
  IPAddress ipAddr = ipHost.AddressList[0];
  IPEndPoint localEndPoint = new IPEndPoint(ipAddr, 11111);

  // Creation TCP/IP Socket using
  // Socket Class Constructor
  Socket listener = new Socket(ipAddr.AddressFamily,
        SocketType.Stream, ProtocolType.Tcp);
```



```
try {

  // Using Bind() method we associate a
  // network address to the Server Socket
  // All client that will connect to this
  // Server Socket must know this network
  // Address
  listener.Bind(localEndPoint);

  // Using Listen() method we create
  // the Client list that will want
  // to connect to Server
  listener.Listen(10);

  while (true) {

    Console.WriteLine("Waiting connection ... ");

    // Suspend while waiting for
    // incoming connection Using
    // Accept() method the server
    // will accept connection of client
    Socket clientSocket = listener.Accept();

    // Data buffer
    byte[] bytes = new Byte[1024];
    string data = null;

    while (true) {
```



```
        int numByte = clientSocket.Receive(bytes);

        data += Encoding.ASCII.GetString(bytes,
                    0, numByte);

        if (data.IndexOf("<EOF>") > -1)
          break;
      }

      Console.WriteLine("Text received -> {0} ", data);
      byte[] message = Encoding.ASCII.GetBytes("Test Server");

      // Send a message to Client
      // using Send() method
      clientSocket.Send(message);

      // Close client Socket using the
      // Close() method. After closing,
      // we can use the closed Socket
      // for a new Client Connection
      clientSocket.Shutdown(SocketShutdown.Both);
      clientSocket.Close();
    }
  }

  catch (Exception e) {
    Console.WriteLine(e.ToString());
  }
}
```

NetworkStreamClass

The NetworkStream class provides methods for sending and receiving data over Stream sockets in blocking mode, To create a NetworkStream, you must provide a connected Socket. You can also specify what FileAccess permission the NetworkStream has over the provided Socket.

Namespace: System.Net.Sockets

Properties

CanRead	GetsavaluethatindicateswhethertheNetworkStream supportsreading.
CanWrite	GetsavaluethatindicateswhethertheNetworkStream supportswriting.
Length	Getsthelengthofthedataavailableonthestream.
Socket	Getstheunderlying Socket.

Methods

Read(Byte[], Int32, Int32)	ReadsdatafromtheNetworkStreamandstoresittoa bytearray.
Write(Byte[], Int32, Int32)	WritesdatatotheNetworkStreamfromaspecifiedrange ofabytearray.

EncodingClass

EncodingistheprocessoftransformingasetofUnicodecharactersintoasequence ofbytes.Incontrast,decodingistheprocessoftransformingasequenceofencoded bytes into a set of Unicode characters.

.NET provides the following implementations of the Encoding class to support current Unicode encodings and other encodings:

- ASCIIEncodingencodesUnicodecharactersassingle7-bitASCIIcharacters.
- UTF8EncodingencodesUnicodecharactersusingtheUTF-8encoding.
- UnicodeEncodingencodesUnicodecharactersusingtheUTF-16encoding.

Namespace:System.Text

Methods

GetBytes(Char[])	When overridden in a derived class,encodes all the charactersinthespecifiedcharacterarrayintoa sequenceofbytes.
GetBytes(String)	When overridden in a derived class, encodes all the charactersinthespecifiedstringintoasequenceof bytes.
GetString(Byte[])	Whenoverriddeninaderivedclass,decodesallthe bytesinthespecifiedbytearrayintoastring.
ToString()	Returnsastringthatrepresentsthecurrentobject.

TcpListenerClass

The TcpListener class provides simple methods that listen for and accept incoming connectionrequestsinblockingsynchronousmode.You canuseeitheraTcpClient or a Socket to connect with a TcpListener. Create a TcpListener using an IPEndPoint, a Local IP address and port number, or just a port number.

Use the Start method to begin listening for incoming connection requests. Startwillqueueincomingconnectionsuntilyoueithercallthe Stopmethodorithas queued MaxConnections. Use either AcceptSocket or AcceptTcpClient to pull a connection from the incoming connection request queue.

Namespace: System.Net.Sockets

Methods

AcceptTcpClient()	Acceptsapendingconnectionrequest.
Pending()	Determinesiftherearependingconnectionrequests.
Start()	Startslisteningforincomingconnectionrequests.
Stop()	Closesthelistener.
ToString()	Returnsastringthatrepresentsthecurrentobject.
Create(Int32)	CreatesanewTcpListenerinstancetolistenonthe specified port.

AcceptSocket()	Acceptsapendingconnectionrequest.

Properties

Active	GetsavaluethatindicateswhetherTcpListeneris actively listening for client connections.
LocalEndpoint	GetstheunderlyingEndPointofthecurrentTcpListener.
Server	GetstheunderlyingnetworkSocket.

TcpClientClass

TheTcpClientclassprovidessimplemethodsforconnecting,sending,andreceiving stream data over a network in synchronous blocking mode.

InorderforTcpClienttoconnectandexchangedata,aTcpListenerorSocketcreated withtheTCPProtocolTypemustbelisteningforincomingconnectionrequests.You can connect to this listener in one of the following two ways:

- CreateaTcpClientandcalloneofthethreeavailableConnect methods.
- CreateaTcpClientusingthehostnameandportnumberoftheremotehost. This constructor will automatically attempt a connection.

To send and receive data, use the GetStream() method to obtain a NetworkStream. Call the Write(Byte[], Int32, Int32) and Read(Byte[], Int32, Int32) methods of the NetworkStreamtosendandreceivedatawiththeremotehost.UsetheClose(Int32) method to release all resources associated with the TcpClient.

Namespace: System.Net.Sockets

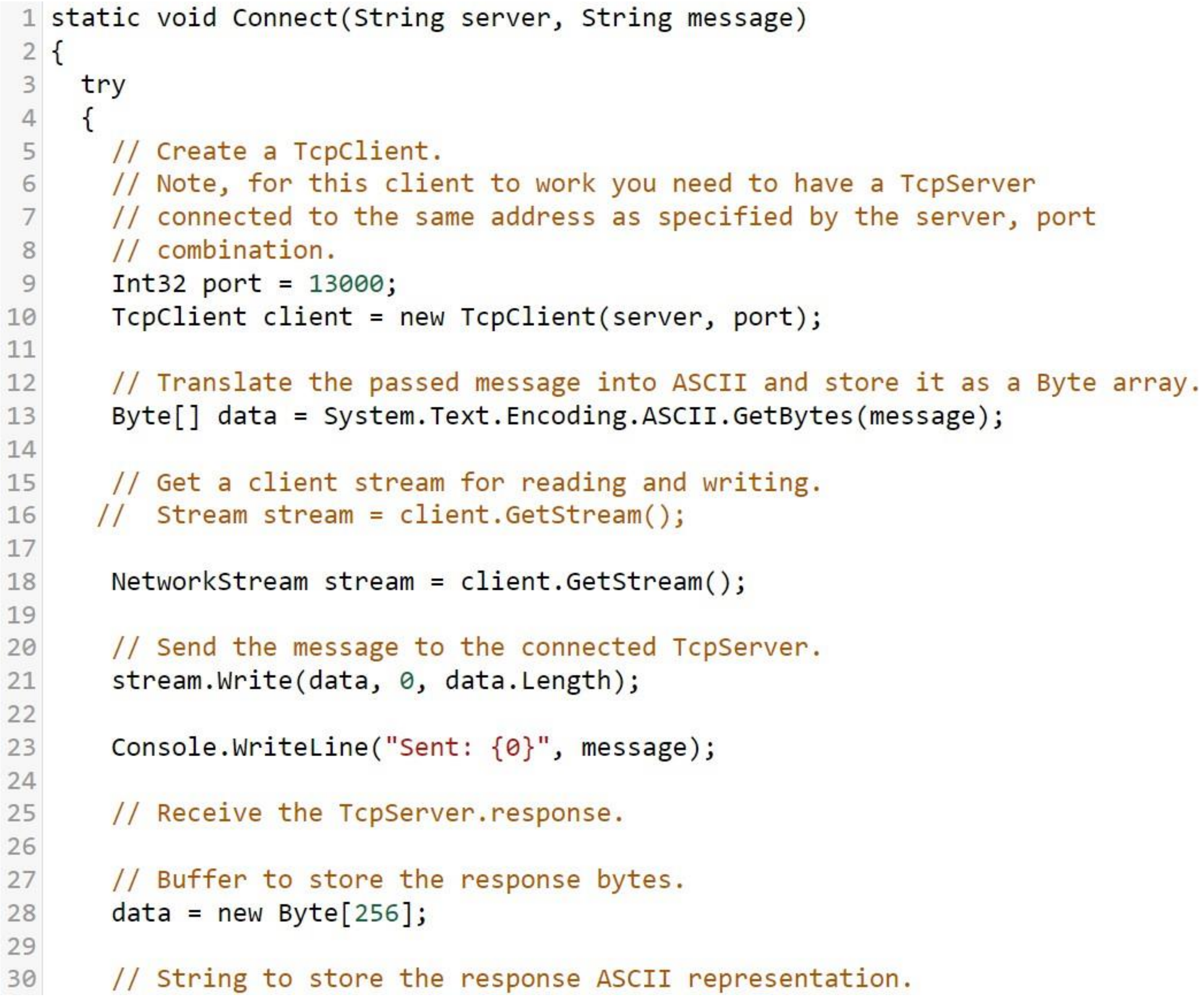
Methods

Connect(IPAddress,Int32)	ConnectstheclienttoaremoteTCPhostusingthe specified IP address and port number.
Connect(IPEndPoint)	ConnectstheclienttoaremoteTCPhostusingthe specified remote network endpoint.

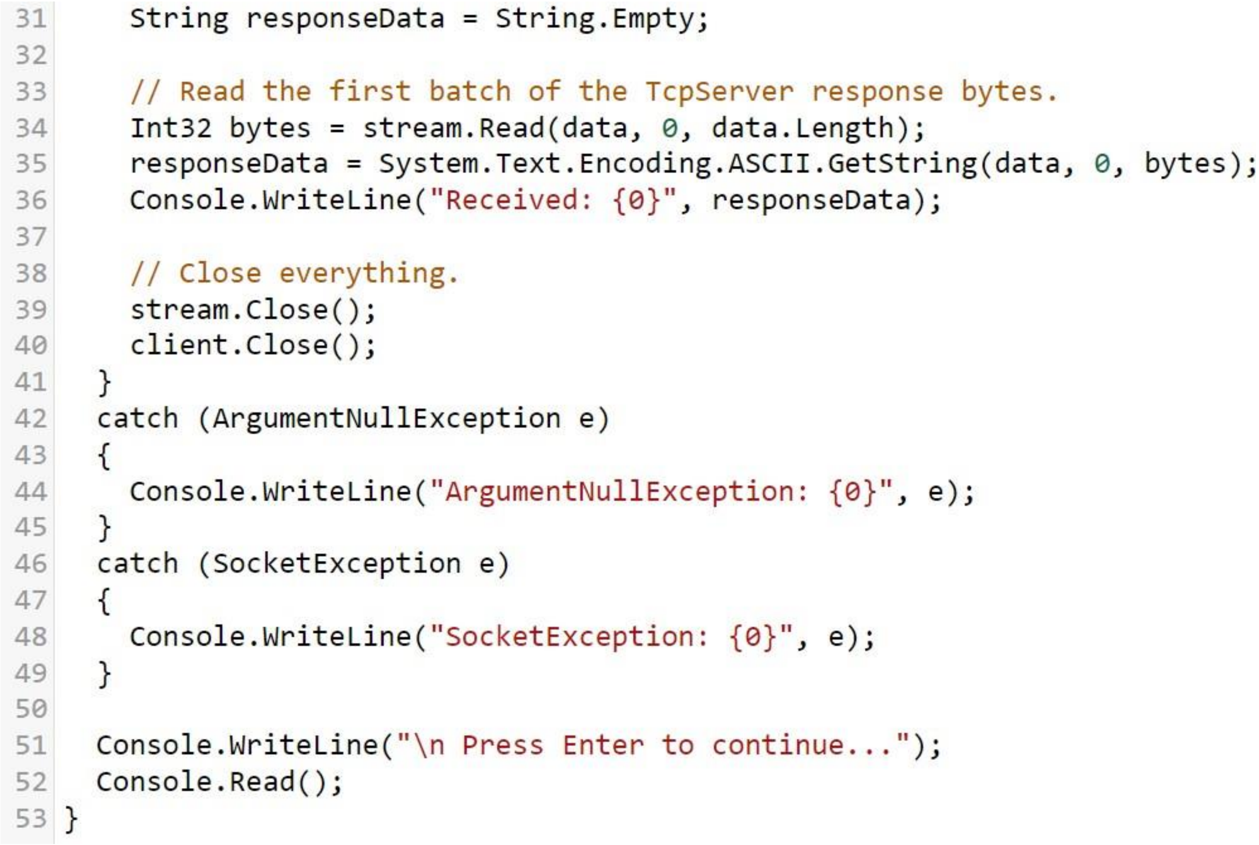
Dispose()	Releasesthemanagedandunmanagedresources used by the TcpClient.
GetStream()	ReturnstheNetworkStreamusedtosendand receive data.
ToString()	Returnsastringthatrepresentsthecurrentobject.
Close()	DisposesthisTcpClientinstanceandrequeststhat the underlying TCP connection be closed.

Properties

Active	Getsorsetsavaluethatindicateswhethera connection has been made.
Connected	Gets a value indicating whether the underlying SocketforaTcpClientisconnectedtoaremote host.



```
static void Connect(String server, String message)
{
  try
  {
    // Create a TcpClient.
    // Note, for this client to work you need to have a TcpServer
    // connected to the same address as specified by the server, port
    // combination.
    Int32 port = 13000;
    TcpClient client = new TcpClient(server, port);

    // Translate the passed message into ASCII and store it as a Byte array.
    Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

    // Get a client stream for reading and writing.
   //  Stream stream = client.GetStream();

    NetworkStream stream = client.GetStream();

    // Send the message to the connected TcpServer.
    stream.Write(data, 0, data.Length);

    Console.WriteLine("Sent: {0}", message);

    // Receive the TcpServer.response.

    // Buffer to store the response bytes.
    data = new Byte[256];

    // String to store the response ASCII representation.
```



```
    String responseData = String.Empty;

    // Read the first batch of the TcpServer response bytes.
    Int32 bytes = stream.Read(data, 0, data.Length);
    responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes);
    Console.WriteLine("Received: {0}", responseData);

    // Close everything.
    stream.Close();
    client.Close();
  }
  catch (ArgumentNullException e)
  {
    Console.WriteLine("ArgumentNullException: {0}", e);
  }
  catch (SocketException e)
  {
    Console.WriteLine("SocketException: {0}", e);
  }

  Console.WriteLine("\n Press Enter to continue...");
  Console.Read();
}
```