**Using an Asynchronous Server Socket**

**.NET Framework 4**

Asynchronous server sockets use the .NET Framework asynchronous programming model to process network service requests. The [Socket](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.aspx) class follows the standard .NET Framework asynchronous naming pattern; for example, the synchronous [Accept](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.accept.aspx) method corresponds to the asynchronous [BeginAccept](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.beginaccept.aspx) and [EndAccept](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.endaccept.aspx) methods.

An asynchronous server socket requires a method to begin accepting connection requests from the network, a callback method to handle the connection requests and begin receiving data from the network, and a callback method to end receiving the data. All these methods are discussed further in this section.

In the following example, to begin accepting connection requests from the network, the method StartListening initializes the **Socket** and then uses the **BeginAccept** method to start accepting new connections. The accept callback method is called when a new connection request is received on the socket. It is responsible for getting the **Socket** instance that will handle the connection and handing that **Socket** off to the thread that will process the request. The accept callback method implements the [AsyncCallback](http://msdn.microsoft.com/en-us/library/system.asynccallback.aspx) delegate; it returns a void and takes a single parameter of type [IAsyncResult](http://msdn.microsoft.com/en-us/library/system.iasyncresult.aspx). The following example is the shell of an accept callback method.

**C#**

void acceptCallback( IAsyncResult ar) {

// Add the callback code here.

}

The **BeginAccept** method takes two parameters, an **AsyncCallback** delegate that points to the accept callback method and an object that is used to pass state information to the callback method. In the following example, the listening **Socket** is passed to the callback method through the *state* parameter. This example creates an **AsyncCallback** delegate and starts accepting connections from the network.

**C#**

listener.BeginAccept(

new AsyncCallback(SocketListener.acceptCallback),

listener);

Asynchronous sockets use threads from the system thread pool to process incoming connections. One thread is responsible for accepting connections, another thread is used to handle each incoming connection, and another thread is responsible for receiving data from the connection. These could be the same thread, depending on which thread is assigned by the thread pool. In the following example, the [System.Threading.ManualResetEvent](http://msdn.microsoft.com/en-us/library/system.threading.manualresetevent.aspx) class suspends execution of the main thread and signals when execution can continue.

The following example shows an asynchronous method that creates an asynchronous TCP/IP socket on the local computer and begins accepting connections. It assumes that there is a global **ManualResetEvent** named allDone, that the method is a member of a class named SocketListener, and that a callback method named acceptCallback is defined.

**C#**

public void StartListening() {

IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName());

IPEndPoint localEP = new IPEndPoint(ipHostInfo.AddressList[0],11000);

Console.WriteLine("Local address and port : {0}",localEP.ToString());

Socket listener = new Socket( localEP.Address.AddressFamily,

SocketType.Stream, ProtocolType.Tcp );

try {

listener.Bind(localEP);

listener.Listen(10);

while (true) {

allDone.Reset();

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

listener.BeginAccept(

new AsyncCallback(SocketListener.acceptCallback),

listener );

allDone.WaitOne();

}

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

Console.WriteLine( "Closing the listener...");

}

The accept callback method (acceptCallback in the preceding example) is responsible for signaling the main application thread to continue processing, establishing the connection with the client, and starting the asynchronous read of data from the client. The following example is the first part of an implementation of the acceptCallback method. This section of the method signals the main application thread to continue processing and establishes the connection to the client. It assumes a global **ManualResetEvent** named allDone.

**C#**

public void acceptCallback(IAsyncResult ar) {

allDone.Set();

Socket listener = (Socket) ar.AsyncState;

Socket handler = listener.EndAccept(ar);

// Additional code to read data goes here.

}

Reading data from a client socket requires a state object that passes values between asynchronous calls. The following example implements a state object for receiving a string from the remote client. It contains fields for the client socket, a data buffer for receiving data, and a [StringBuilder](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.aspx) for creating the data string sent by the client. Placing these fields in the state object allows their values to be preserved across multiple calls to read data from the client socket.

**C#**

public class StateObject {

public Socket workSocket = null;

public const int BufferSize = 1024;

public byte[] buffer = new byte[BufferSize];

public StringBuilder sb = new StringBuilder();

}

The section of the acceptCallback method that starts receiving the data from the client socket first initializes an instance of the StateObject class and then calls the [BeginReceive](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.beginreceive.aspx) method to start reading the data from the client socket asynchronously.

The following example shows the complete acceptCallback method. It assumes that there is a global **ManualResetEvent** named allDone, that the StateObject class is defined, and that the readCallback method is defined in a class named SocketListener.

**C#**

public static void acceptCallback(IAsyncResult ar) {

// Get the socket that handles the client request.

Socket listener = (Socket) ar.AsyncState;

Socket handler = listener.EndAccept(ar);

// Signal the main thread to continue.

allDone.Set();

// Create the state object.

StateObject state = new StateObject();

state.workSocket = handler;

handler.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,

new AsyncCallback(AsynchronousSocketListener.readCallback), state);

}

The final method that needs to be implemented for the asynchronous socket server is the read callback method that returns the data sent by the client. Like the accept callback method, the read callback method is an **AsyncCallback** delegate. This method reads one or more bytes from the client socket into the data buffer and then calls the **BeginReceive** method again until the data sent by the client is complete. Once the entire message has been read from the client, the string is displayed on the console and the server socket handling the connection to the client is closed.

The following sample implements the readCallback method. It assumes that the StateObject class is defined.

**C#**

public static void readCallback(IAsyncResult ar) {

StateObject state = (StateObject) ar.AsyncState;

Socket handler = state.WorkSocket;

// Read data from the client socket.

int read = handler.EndReceive(ar);

// Data was read from the client socket.

if (read > 0) {

state.sb.Append(Encoding.ASCII.GetString(state.buffer,0,read));

handler.BeginReceive(state.buffer,0,StateObject.BufferSize, 0,

new AsyncCallback(readCallback), state);

} else {

if (state.sb.Length > 1) {

// All the data has been read from the client;

// display it on the console.

string content = state.sb.ToString();

Console.WriteLine("Read {0} bytes from socket.\n Data : {1}",

content.Length, content);

}

handler.Close();

}

}