**Using an Asynchronous Client Socket**

**.NET Framework 4**

* [.NET Framework 4.5](http://msdn.microsoft.com/en-us/library/bbx2eya8(v=vs.110).aspx)
* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/bbx2eya8(v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/bbx2eya8(v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/bbx2eya8(v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/bbx2eya8(v=vs.71).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/bbx2eya8(v=vs.90).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/bbx2eya8(v=vs.80).aspx)

An asynchronous client socket does not suspend the application while waiting for network operations to complete. Instead, it uses the standard .NET Framework asynchronous programming model to process the network connection on one thread while the application continues to run on the original thread. Asynchronous sockets are appropriate for applications that make heavy use of the network or that cannot wait for network operations to complete before continuing.

The [Socket](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.aspx) class follows the .NET Framework naming pattern for asynchronous methods; for example, the synchronous [Receive](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.receive.aspx) method corresponds to the asynchronous [BeginReceive](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.beginreceive.aspx) and [EndReceive](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.endreceive.aspx) methods.

Asynchronous operations require a callback method to return the result of the operation. If your application does not need to know the result, then no callback method is required. The example code in this section demonstrates using a method to start connecting to a network device and a callback method to complete the connection, a method to start sending data and a callback method to complete the send, and a method to start receiving data and a callback method to end receiving data.

Asynchronous sockets use multiple threads from the system thread pool to process network connections. One thread is responsible for initiating the sending or receiving of data; other threads complete the connection to the network device and send or receive the data. In the following examples, instances of the [System.Threading.ManualResetEvent](http://msdn.microsoft.com/en-us/library/system.threading.manualresetevent.aspx) class are used to suspend execution of the main thread and signal when execution can continue.

In the following example, to connect an asynchronous socket to a network device, the Connect method initializes a **Socket** and then calls the [BeginConnect](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.connect.aspx) method, passing a remote endpoint that represents the network device, the connect callback method, and a state object (the client **Socket**), which is used to pass state information between asynchronous calls. The example implements the Connect method to connect the specified **Socket** to the specified endpoint. It assumes a global **ManualResetEvent** named connectDone.

**C#**

public static void Connect(EndPoint remoteEP, Socket client) {

client.BeginConnect(remoteEP,

new AsyncCallback(ConnectCallback), client );

connectDone.WaitOne();

}

The connect callback method ConnectCallback implements the [AsyncCallback](http://msdn.microsoft.com/en-us/library/system.asynccallback.aspx) delegate. It connects to the remote device when the remote device is available and then signals the application thread that the connection is complete by setting the **ManualResetEvent** connectDone. The following code implements the ConnectCallback method.

**C#**

private static void ConnectCallback(IAsyncResult ar) {

try {

// Retrieve the socket from the state object.

Socket client = (Socket) ar.AsyncState;

// Complete the connection.

client.EndConnect(ar);

Console.WriteLine("Socket connected to {0}",

client.RemoteEndPoint.ToString());

// Signal that the connection has been made.

connectDone.Set();

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

The example method Send encodes the specified string data in ASCII format and sends it asynchronously to the network device represented by the specified socket. The following example implements the Send method.

**C#**

private static void Send(Socket client, String data) {

// Convert the string data to byte data using ASCII encoding.

byte[] byteData = Encoding.ASCII.GetBytes(data);

// Begin sending the data to the remote device.

client.BeginSend(byteData, 0, byteData.Length, SocketFlags.None,

new AsyncCallback(SendCallback), client);

}

The send callback method SendCallback implements the [AsyncCallback](http://msdn.microsoft.com/en-us/library/system.asynccallback.aspx) delegate. It sends the data when the network device is ready to receive. The following example shows the implementation of the SendCallback method. It assumes a global **ManualResetEvent** named sendDone.

**C#**

private static void SendCallback(IAsyncResult ar) {

try {

// Retrieve the socket from the state object.

Socket client = (Socket) ar.AsyncState;

// Complete sending the data to the remote device.

int bytesSent = client.EndSend(ar);

Console.WriteLine("Sent {0} bytes to server.", bytesSent);

// Signal that all bytes have been sent.

sendDone.Set();

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

Reading data from a client socket requires a state object that passes values between asynchronous calls. The following class is an example state object for receiving data from a client socket. It contains a field for the client socket, a buffer for the received data, and a [StringBuilder](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.aspx) to hold the incoming data string. 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 {

// Client socket.

public Socket workSocket = null;

// Size of receive buffer.

public const int BufferSize = 256;

// Receive buffer.

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

// Received data string.

public StringBuilder sb = new StringBuilder();

}

The example Receive method sets up the state object and then calls the **BeginReceive** method to read the data from the client socket asynchronously. The following example implements the Receive method.

**C#**

private static void Receive(Socket client) {

try {

// Create the state object.

StateObject state = new StateObject();

state.workSocket = client;

// Begin receiving the data from the remote device.

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

new AsyncCallback(ReceiveCallback), state);

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

The receive callback method ReceiveCallback implements the **AsyncCallback** delegate. It receives the data from the network device and builds a message string. It reads one or more bytes of data from the network into the data buffer and then calls the **BeginReceive** method again until the data sent by the client is complete. Once all the data is read from the client, ReceiveCallback signals the application thread that the data is complete by setting the **ManualResetEvent** sendDone.

The following example code implements the ReceiveCallback method. It assumes a global string named response that holds the received string and a global **ManualResetEvent** named receiveDone. The server must shut down the client socket gracefully to end the network session.

**C#**

private static void ReceiveCallback( IAsyncResult ar ) {

try {

// Retrieve the state object and the client socket

// from the asynchronous state object.

StateObject state = (StateObject) ar.AsyncState;

Socket client = state.workSocket;

// Read data from the remote device.

int bytesRead = client.EndReceive(ar);

if (bytesRead > 0) {

// There might be more data, so store the data received so far.

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

// Get the rest of the data.

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

new AsyncCallback(ReceiveCallback), state);

} else {

// All the data has arrived; put it in response.

if (state.sb.Length > 1) {

response = state.sb.ToString();

}

// Signal that all bytes have been received.

receiveDone.Set();

}

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}