Asynchronous File Operations

The great thing about asynchronous support in .NET is that once you become familiar with its programming pattern, you can apply it to several classes that expose asynchronous operations natively—that is, without the need of an explicit asynchronous delegate to the method. In this section, I'll show you how to use the BeginRead, EndRead, BeginWrite, and EndWrite methods of the Stream class to perform asynchronous file I/O. All the stream-based classes, including FileStream, inherit these methods.

The BeginWrite method takes a Byte array that contains the data to be written to the stream, the index of the first element to write, and the number of bytes to write: these are the same arguments that the regular, synchronous Write method accepts. You also pass an AsyncCallback delegate and a state object, as you do with all the asynchronous method invocations that you've seen in earlier sections. The callback routine must conclude the write operation by invoking the EndWrite method and then close the stream.

The BeginRead method has the same argument signature as BeginWrite, with the first three values defining the location at which data read from the stream will be stored. The callback routine must conclude the asynchronous read operation by invoking an EndRead method and then close the stream. The EndRead method returns the total number of bytes read; a 0 value means that there were no more bytes to read.

The following code shows an example of asynchronous write and read operations on the same file. A single callback routine serves both the write and the read operation: the type of operation is passed as a string in the last argument to BeginWrite and BeginRead. To keep the code simple, both the caller routine and the callback routine share the variable pointing to the Byte array buffer and the FileStream object. In a real-world application, you might want to pack this data into an object and pass it as the last argument to BeginWrite and BeginRead:

```
' The file being read from or written to
Const FileName As String = "C:\TESTDATA.TMP"
' The FileStream object used for both reading and writing
Dim fs As System.IO.FileStream
' The buffer for file I/O
Dim buffer() As Byte

' This procedure tests asynchronous file read.

Sub TestAsyncFileOperations()
    ' Fill the buffer with 20 KB of random data.
    ReDim buffer(1048575)
    For i As Integer = 0 To UBound(buffer)
        buffer(i) = CByte(i Mod 256)
    Next

    ' Create the target file in asynchronous mode (open in
    ' asynchronous mode).
    fs = New System.IO.FileStream(FileName, IO.FileMode.Create, _
        IO.FileAccess.Write, IO.FileShare.None, 65536, True)
    ' Start the async write operation.
    Console.WriteLine("Starting the async write operation")
    Dim ar As IAsyncResult = fs.BeginWrite(buffer, 0, _
```

```
            UBound(buffer) + 1, AddressOf AsyncFileCallback, "write")

        ' Wait a few seconds until the operation completes.
        Thread.Sleep(4000)
        ' Now read the file back.
        fs = New System.IO.FileStream(FileName, IO.FileMode.Open, _
            IO.FileAccess.Read, IO.FileShare.None, 65536, True)
        ' Size the receiving buffer.
        ReDim buffer(CInt(fs.Length) - 1)
        ' Start the async read operation.
        Console.WriteLine("Starting the async read operation")
        ar = fs.BeginRead(buffer, 0, UBound(buffer) + 1, _
            AddressOf AsyncFileCallback, "read")
    End Sub

    ' This is the callback procedure for both async read and write.

    Sub AsyncFileCallback(ByVal ar As IAsyncResult)
        ' Get the state object (the "write" or "read" string).
        Dim opName As String = ar.AsyncState.ToString

        ' The behavior is quite different in the two cases.
        Select Case opName
            Case "write"
                Console.WriteLine("Async write operation completed")
                ' Complete the write, and close the stream.
                fs.EndWrite(ar)
                fs.Close()
            Case "read"
                Console.WriteLine("Async read operation completed")
                ' Complete the read, and close the stream.
                Dim bytes As Integer = fs.EndRead(ar)
                Console.WriteLine("Read {0} bytes", bytes)
                fs.Close()
        End Select
    End Sub
```

You get the best benefits from asynchronous file I/O if you also open the FileStream for asynchronous operations by passing True in the last argument of the object's constructor:

```
fs = New FileStream(path, mode, access, share, bufferSize,
useAsync)
```

When you open a FileStream in this way, synchronous operations are slowed down, but asynchronous operations are completed faster. Keep in mind that read and write operations of less then 64 KB are usually performed synchronously anyway, even if you use BeginWrite or BeginRead, and that the useAsync argument might be ignored on Windows platforms that don't support asynchronous file operations. You can test whether the FileStream was actually opened for asynchronous operation by testing its IsAsync property.