Adam O'Neil's Development Blog

Streaming Files (for Upload/Download) in WCF (Message Contracts)

I recently had to write some code to perform an upload to a WCF service, and there was a chance that the files could be a touch on the large side so streaming seemed like the best option.

There is quite a limited amount of information about this subject – and configuring the web config is a bit tricky, so I have posted some examples of how to do this in the hope that someone will find it useful..

You need to start by defining some message contracts for your upload / download.. these need to be defined in the interface for your WCF service (the file that contains the definitions and contracts for your service) :-

```
[MessageContract]
public class FileUploadMessage
    {
        [MessageHeader(MustUnderstand = true)]
        public PublishingMetaData Metadata;
        [MessageHeader(MustUnderstand = true)]
        public string AuthenticationKey;
        [MessageBodyMember(Order = 1)]
        public Stream FileByteStream;
    }

    [MessageContract]
    public class FileDownloadMessage
    {
        [MessageHeader(MustUnderstand = true)]
        public PublishingMetaData FileMetaData;
        [MessageHeader(MustUnderstand = true)]
        public string AuthenticationKey;
    }

[MessageContract]
    public class FileDownloadReturnMessage
    {
        public FileDownloadReturnMessage(PublishingMetaData metaData, Stream stream)
        {
            this.DownloadedFileMetadata = metaData;
            this.FileByteStream = stream;
        }

        [MessageHeader(MustUnderstand = true)]
        public PublishingMetaData DownloadedFileMetadata;
        [MessageBodyMember(Order = 1)]
        public Stream FileByteStream;
    }
```

Notice that for the FileUploadMessage I have included a Stream.. the PublishingMetaData and AuthenticationKey are custom classes / properties and don't need to be implemented in your version.

I also need to define a couple of web methods in the interface which are used for uploading / downloading files :-

```
[OperationContract(IsOneWay = false)]
        FileDownloadReturnMessage DownloadFile(FileDownloadMessage request);

[OperationContract(IsOneWay = true)]
        void UploadFile(FileUploadMessage request);

[OperationContract]
        void AttemptToCloseStream(string authenticationKey, PublishingMetaData metaData);
```

Now we have defined our contracts – here is the implementation (which is contained in the main WCF service class).. I have left my security checking and various other custom code in for illustration purposes, but again this can be removed in your implementation :-

```
public void UploadFile(FileUploadMessage request)
        {
            if (!CheckAuthenticationKey(request.AuthenticationKey)) { throw new SecurityException("The user do
            Stream fileStream = null;
            Stream outputStream = null;

            try
            {
                fileStream = request.FileByteStream;

                string rootPath = ConfigurationManager.AppSettings["RootPath"].ToString();

                DirectoryInfo dirInfo = new DirectoryInfo(rootPath);
                if (!dirInfo.Exists)
                {
                    dirInfo.Create();
                }

                //Create the file in the filesystem - change the extension if you wish, or use a passed in valu
                string newFileName = Path.Combine(rootPath, Guid.NewGuid() + ".xml");

                outputStream = new FileInfo(newFileName).OpenWrite();
                const int bufferSize = 1024;
                byte[] buffer = new byte[bufferSize];

                int bytesRead = fileStream.Read(buffer, 0, bufferSize);

                while (bytesRead &gt; 0)
                {
                    outputStream.Write(buffer, 0, bufferSize);
                    bytesRead = fileStream.Read(buffer, 0, bufferSize);
                }
            }
            catch (IOException ex)
            {
                throw new FaultException&lt;IOException&gt;(ex, new FaultReason(ex.Message));
            }
            finally
            {
                if (fileStream != null)
                {
                    fileStream.Close();
                }
                if (outputStream != null)
                {
                    outputStream.Close();
                }
            }
        }
```

And here is my download implementation (notice the use of a list of OpenStreams.. this is a workaround to fix a problem I was having with streams being left open .. I use this to allow my program to call the service and ensure the stream is closed after the file is downloaded) :-

```
static Dictionary&lt;string, Stream&gt; OpenStreams { get; set; }
```

```

public FileDownloadReturnMessage DownloadFile(FileDownloadMessage request)
        {
            try
            {
                if (!CheckAuthenticationKey(request.AuthenticationKey)) { throw new SecurityException("The use
                string rootPath = ConfigurationManager.AppSettings["RootPath"].ToString();
                Stream fileStream = new FileStream(Path.Combine(rootPath, Path.GetFileName(request.FileMetaDat
                if (ExecutionResearchService.OpenStreams == null)
                {
                    ExecutionResearchService.OpenStreams = new Dictionary&lt;string, Stream&gt;();
                }
                ExecutionResearchService.OpenStreams.Add(Path.GetFileName(request.FileMetaData.FileName), file
                return new FileDownloadReturnMessage(new PublishingMetaData(), fileStream);
            }
            catch (IOException ex)
            {
                throw new FaultException&lt;IOException&gt;(ex, new FaultReason(ex.Message));
            }
        }

public void AttemptToCloseStream(string authenticationKey, PublishingMetaData metaData)
        {
            if (!CheckAuthenticationKey(authenticationKey)) { throw new SecurityException("The user does not h
            if (ExecutionResearchService.OpenStreams != null)
            {
                if (ExecutionResearchService.OpenStreams.ContainsKey(Path.GetFileName(metaData.FileName)))
                {
                    Stream stream = ExecutionResearchService.OpenStreams[Path.GetFileName(metaData.FileName)];
                    stream.Flush();
                    stream.Close();
                    OpenStreams.Remove(Path.GetFileName(metaData.FileName));
                }
            }
        }
```

OK – so now we have a file upload method, download method, and the required contracts we need. The services section of the web config looks like this :-

```
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
```

```
        <behavior name="serviceBehavior">
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="true" httpHelpPageEnabled="true" />
          <dataContractSerializer maxItemsInObjectGraph="2147483647"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <!--http://services.myserviceaddress.com/service.svc-->
      <service behaviorConfiguration="serviceBehavior" name="Projects.MyServiceName">
        <endpoint address="http://services.myserviceaddress.com/service.svc"
        name="basicHttpStream"
        binding="basicHttpBinding"
        bindingConfiguration="httpLargeMessageStream"
        contract="Projects.IMyServiceInterface" />
        <host>
          <baseAddresses>
            <add baseAddress="http://services.myserviceaddress.com/service.svc" />
            <!--<add baseAddress="http://localhost/ExecutionResearchService/ExecutionResearchService.svc" />--
          </baseAddresses>
        </host>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
      </service>
    </services>
    <bindings>
      <basicHttpBinding>
        <binding name="httpLargeMessageStream"
                  maxReceivedMessageSize="2147483647"
                  transferMode="Streamed"
                  messageEncoding="Mtom" />
      </basicHttpBinding>
    </bindings>
  </system.serviceModel>
```

The important bits are the binding section at the bottom – transferMode = “Streamed” and messageEncoding = “Mtom” .. also I have set the maxReceivedMessageSize to it’s maximum value to ensure I can transfer massive files across my web service without issues.

Now – once these are set up and working in your WCF Service – we can add a reference to it and call the methods using our client application.. here is some code on how to do this too, because I found help lacking in this area also!

This is how we upload a file – please change variables, and remove AuthenticationKey and the MetaData objects if you didn’t use them.

```
using (ResearchServiceClient client = WebServiceProxy.GetResearchServiceClient())
{
    Stream fileStream = null;

    try
    {
        string rootPath = @"C:\MyRootFolder";
        string localDocumentPath = Path.Combine(rootPath, "MyNewFileName.xml");
        fileStream = new FileInfo(localDocumentPath).OpenRead();
        client.UploadFile(WebServiceProxy.AuthenticationKey, LivePaths.WorkingPublishingMetaData, fileStream);

        byte[] buffer = new byte[2048];
        int bytesRead = fileStream.Read(buffer, 0, 2048);
        while (bytesRead &gt; 0)
        {
            fileStream.Write(buffer, 0, 2048);
            bytesRead = fileStream.Read(buffer, 0, 2048);
        }
    }
    catch
    {
        throw;
    }
    finally
    {
        if (fileStream != null)
        {
            fileStream.Close();
        }
    }
}

using (ResearchServiceClient client = WebServiceProxy.GetResearchServiceClient())
{
    Stream fileStream = null;
    client.DownloadFile(WebServiceProxy.AuthenticationKey, metaData, out fileStream);

    Stream outputStream = null;

    try
    {
        outputStream = new FileInfo("PathForLocalDocument.xml").OpenWrite();
```

```
            byte[] buffer = new byte[2048];

            int bytesRead = fileStream.Read(buffer, 0, 2048);

            while (bytesRead &gt; 0)
            {
                outputStream.Write(buffer, 0, 2048);
                bytesRead = fileStream.Read(buffer, 0, 2048);
            }
        }
        catch
        {

        }
        finally
        {
            if (fileStream != null)
            {
                fileStream.Close();
            }
            if (outputStream != null)
            {
                outputStream.Close();
            }
            client.AttemptToCloseStream(WebServiceProxy.AuthenticationKey, metaData);
        }
    }
```

So – here’s the total solution, and notice we put the client.AttemptToCloseStream in the finally section of our Try / Catch which attempts to close the download stream when we have finished with it. It seems like a little bit of a hack, but I scratched my head trying to find a solution to this, and this is the best thing I could come up with.. it works, so it isn’t that bad.

Comments closed | Trackbacks closed