XML Web Services

MSDN
M. MacDonald, M. Szpuszta, Pro ASP.NET 3.5 in C# 2008. Includes Silverlight 2, 3rd Ed., 2009, Apress

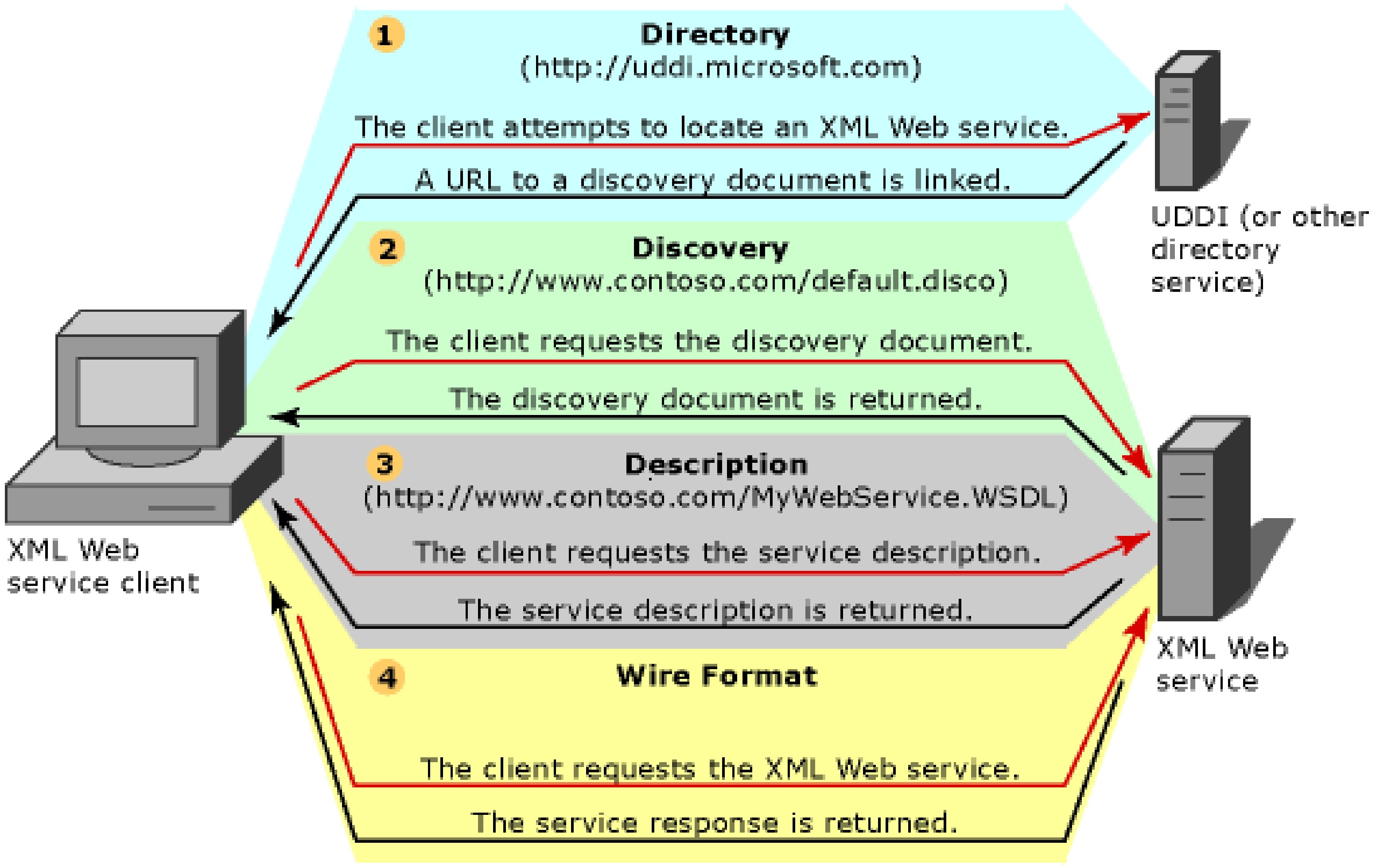
Contents

XML Web Services Overview

Web Services

- A Web service is an application that exposes a Web-accessible API
  - This application can be invoked programmatically over the Web
- The Web services platform is a set of standards that applications follow to achieve interoperability via the Web
  - Any programming language and any platform can be used to write the Web service, and it is widely accessed according to the Web services standards
- The benefits of Web services:
  - Web services are simple
  - Web services are loosely coupled
  - Web services are stateless
  - Web services are firewall-friendly

XML Web Services Infrastructure


1
Directory
(http://uddi.microsoft.com)
The client attempts to locate an XML Web service.
A URL to a discovery document is linked.
UDDI (or other directory service)
2
Discovery
(http://www.contoso.com/default.disco)
The client requests the discovery document.
The discovery document is returned.
XML Web service client
3
Description
(http://www.contoso.com/MyWebService.WSDL)
The client requests the service description.
The service description is returned.
XML Web service
4
Wire Format
The client requests the XML Web service.
The service response is returned.

XML Web Services Standards

- WSDL – used to create an interface definition for a web service
- SOAP – the message format used to encode information (such as data values) before sending it to a web service
- HTTP – the protocol over which all web service communication takes place.
- DISCO – used to create discovery documents that provide links to multiple web service endpoints
- UDDI (Universal Description, Discovery, and Integration) – a standard for creating business registries that catalogue companies, the web services they provide, and the corresponding URLs for their WSDL contracts

Wire Formats

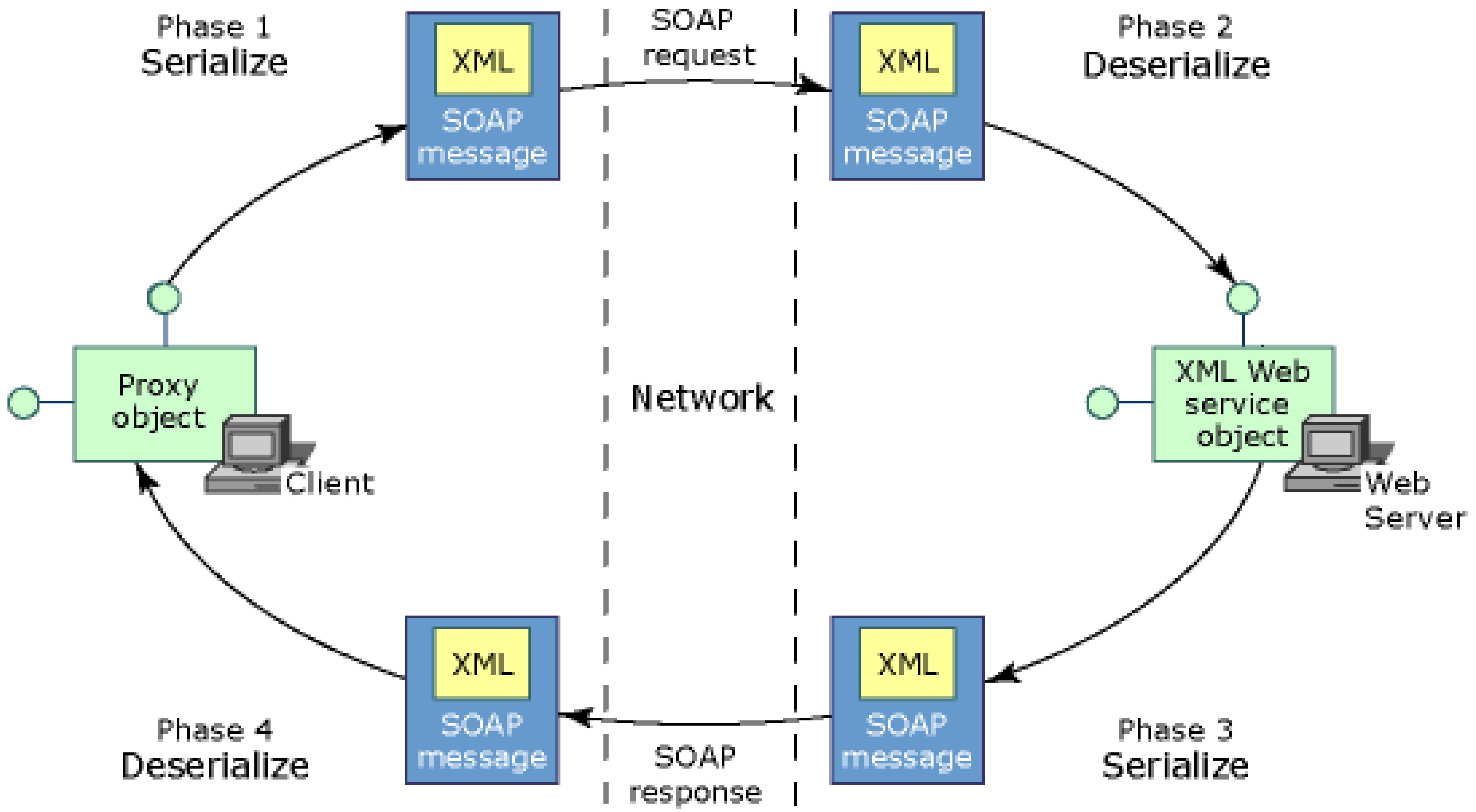
- HTTP-GET and HTTP-POST are standard protocols that use HTTP verbs for the encoding and passing of parameters as name/value pairs
  - HTTP-GET passes its parameters in the form of url encoded text appended to the URL of the server handling the request
  - In HTTP-POST the name/value pairs are passed inside the actual HTTP request message
- SOAP is a simple, lightweight XML-based protocol for exchanging structure and type information on the Web
  - The overall design goal of SOAP is to keep it as simple as possible, and to provide a minimum of functionality
  - The protocol defines a messaging framework that contains no application or transport semantics

SOAP Examples

```xml
<soap:Envelope
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails xmlns="http://warehouse.example.com/ws">
      <productID>827635</productID>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```

```xml
<soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
            <getProductDetailsResult>
                <productName>Toptimate 3-Piece Set</productName>
                <productID>827635</productID>
                <description>3-Piece luggage set.</description>
                <price>96.50</price>
                <inStock>true</inStock>
            </getProductDetailsResult>
        </getProductDetailsResponse>
    </soap:Body>
</soap:Envelope>
```

XML Web Service Lifetime



Creating XML Web Services

Building an XML Web Service

1. Create an .asmx file and declare a Web service using the **`@WebService`** directive
   - This directive specifies the class that implements the Web service and the programming language that is used in the implementation

```
<%@ WebService Language="C#" Class="WebService1" %>
```

   - If the class resides in a separate assembly, it must be placed in the **`\Bin`** directory under the Web application where the Web service resides

```
<%@ WebService Language="C#"
               Class="MyName.WebService1,MyAssembly" %>
```

Building an XML Web Service cont.

2. Create a class that implements the Web service
   - The class can optionally derive from the **`WebService`** class
   - Deriving from the WebService class allows to gain access to the common ASP.NET objects, such as **`Application`**, **`Session`**, **`User`**, and **`Context`**

```
<%@ WebService Language="C#" Class="WebService1" %>

using System;
using System.Web.Services;

public class WebService1 : WebService
```

Building an XML Web Service cont.

3. Optionally, apply the **`WebServiceAttribute`** attribute to the class implementing the Web service
   - It can be used to set the default XML namespace for the Web service
   - The default namespace, **`http://tempuri.org`**, should be changed before the XML Web service is made publicly consumable

```
<%@ WebService Language="C#" Class="WebService1" %>

using System;
using System.Web.Services;

[WebService(Namespace = "http://www.contoso.com/")]
public class WebService1 : WebService
```

Building an XML Web Service cont.

4. Define the Web service methods that compose the functionality of the Web service
   - Web service method is a method that can be communicated with over the Web
   - All Web service methods must be public and have the **`WebMethod`** attribute applied to them

```
<%@ WebService Language="C#" Class="WebService1" %>

using System;
using System.Web.Services;

[WebService(Namespace = "http://www.contoso.com/")]
public class WebService1 : WebService {
    [WebMethod]
    public long Multiply(int a, int b)
    {
        return a * b;
    }
}
```

WebServiceAttribute Members

- **`Description`** – a descriptive message for the Web service
  - Displayed to prospective consumers when description documents are generated
- **`Name`** – the name of the Web service
  - Used in a WSDL document
- **`Namespace`** - used as the default namespace for XML elements directly pertaining to the XML Web service

WebMethodAttribute Members

- **`BufferResponse`** – whether the response for this request is buffered
- **`CacheDuration`** – the number of seconds the response should be held in the cache
- **`Description`** – a descriptive message
- **`EnableSession`** – whether session state is enabled for an XML Web service method
  - In order for an XML Web service to maintain session state for a client, the client must persist the cookie
- **`MessageName`** – can be used to alias method or property names (e.g. to uniquely identify polymorphic methods)
  - The default is the name of the XML Web service method
- **`TransactionOption`** – indicates the transaction support

Asynchronous XML Web Service Methods

- Asynchronous Web service methods should be used to improve performance of Web service methods that invoke long-running methods that block their thread
- Potential reasons for using asynchronous Web methods
  - Communicating with other Web services
  - Accessing remote databases
  - Performing network I/O
  - Reading and writing to large files
- Regardless of whether a Web service method is implemented asynchronously, clients can communicate with it asynchronously
- The implementation of an asynchronous Web service method has no impact on the HTTP connection between the client and the server

Authentication Options

- Windows – Basic: the user name and password are sent in base 64-encoded strings in plain text
- Windows – Basic over SSL: using Secure Sockets Layer
- Windows – Digest: uses hashing to transmit client credentials in an encrypted manner (not supported by other platforms)
- Windows – Integrated Windows: uses a cryptographic exchange with the user's Microsoft Internet Explorer Web browser
- Windows – Client Certificates: requires each client to obtain a certificate from a mutually trusted certificate authority
- Forms: not supported by Web services
- SOAP headers – Custom: user credentials are passed within the SOAP header of the SOAP message

Authentication Using SOAP Headers

- SOAP headers are a great way of passing out-of-band or information not related to the semantics of a Web service
- The `Header` element is optional and can thus be processed by the infrastructure, e.g. to provide custom authentication
- To use SOAP for custom authentication
  - A Web service client would send its credentials to the Web service by adding the expected SOAP header with the client credentials
  - A Web service must do two things:
    - Specify that it expects the SOAP header containing the authentication credentials
    - Authorize the client access to the Web service

Publishing and Deployment XML Web Services

Deploying XML Web Services

- Deploying a Web service involves copying the .asmx file and any assemblies used by the Web service
- The following items are deployed to a Web server:
  - Web application directory – this directory should be flagged as an IIS Web application
  - .asmx file – acts as the base URL for clients
  - .disco file – optional, acts as a discovery mechanism for the Web service
  - Web.config file – optional
  - \Bin directory – contains the binary files for the Web service

Web Service Discovery

- Web service discovery is the process of locating and interrogating Web service descriptions, which is a preliminary step for accessing a Web service
- There are three ways a potential Web service client can access a discovery document:
  - Static discovery file: publish a discovery file, typically with a .disco file name extension
  - ?disco query string: any Web service running on ASP.NET can have a discovery document dynamically generated for it
  - .vsdisco request: dynamic discovery can be turned on to allow Web service client applications to discover all available Web services in the folder and subfolders corresponding to a request URL

Using a .disco File

1. Create and publish a .disco file

```
<?xml version="1.0"?>
<discovery xmlns="http://schemas.xmlsoap.org/disco/">
    <discoveryRef ref="/Folder/Default.disco"/>
    <contractRef ref="http://MyWebServer/UserName.asmx?WSDL"
                docRef="Service.htm"
                 xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
    <schemaRef ref="Schema.xsd"
              xmlns="http://schemas.xmlsoap.org/disco/schema/"/>
</discovery>
```

2. Optionally, create an HTML page with a link to the discovery document
   - Users can then supply URLs like the following during the discovery process

```
<HEAD>
    <link type='text/xml' rel='alternate' href='MyWebService.disco'/>
</HEAD>
```

Enabling Dynamic Discovery

```
<configuration>
    <system.web>
        <httpHandlers>
            <add verb="*" path="*.vsdisco"
     type="System.Web.Services.Discovery.DiscoveryRequestHandler,
                 System.Web.Services, Version=1.0.3300.0,
                 Culture=neutral,
                 PublicKeyToken=b03f5f7f11d50a3a"
                validate="false"/>
        </httpHandlers>
    </system.web>
</configuration>
```

- When dynamic discovery is turned on, all Web services and discovery documents existing on the Web server beneath the requested URL are discoverable

XML Web Services Clients

Web Services Discovery

- When the URL to a discovery document residing on a Web server is known, a developer of a client application can use a Web service discovery to:
  - Learn that a Web service exists
  - Learn what its capabilities are
  - Learn how to properly interact with it
- Through the process of Web service discovery, a set of files is downloaded to the local computer containing details about the existence of Web services
  - Service descriptions, XSD schemas, discovery documents

```
Disco /out:location /username:user /password:mypwd
      /domain:mydomain <url>
```

Creating an XML Web Service Proxy

- As long as a service description exists, a proxy class can be generated if the service description conforms to WSDL
- With a service description, a proxy class can be created using the Wsdl.exe tool

```
Wsdl /language:language  /protocol:protocol /namespace:myNameSpace
     /out:filename /username:username /password:password
     /domain:domain <url or path>
```

  - language: CS, VB, JS, VJS, CPP
  - protocol: SOAP, SOAP 1.2, HTTP-GET, HTTP-POST
  - namespace: the namespace of the generated proxy
  - username and password: used when connecting to a Web server that requires authentication

Generated Proxy Class

- A single source file is generated in the specified language
  - It contains a proxy class exposing both synchronous and asynchronous methods for each Web service method of the Web service
  - Each method of the generated proxy class contains the appropriate code to communicate with the Web service method
  - If an error occurs during communication with the Web service and the proxy class, an exception is thrown

Creating Clients for XML Web Services

1. Create a proxy class for the Web service
2. Reference the proxy class in the client code
3. Create an instance of the proxy class in the client code
4. If anonymous access has been disabled for the Web application hosting the Web service, set the `Credentials` property of the proxy class
5. Call the method on the proxy class that corresponds to the Web service method with which you want to communicate

Asynchronous Communication

- A Web service does not have to be specifically written to handle asynchronous requests to be called asynchronously
- There are two design patterns that allow to call a Web service method asynchronously
  - The Begin/End pattern
    - Using the callback technique
    - Using the wait technique
  - The event-driven asynchronous programming pattern (available in the .NET Framework 2.0)

The Event-Driven Technique

```
private void callButton_Click(object sender, EventArgs e)
{
    BarcodeService.BarcodeSvcSoapClient service = new
        BarcodeClient.BarcodeService.BarcodeSvcSoapClient();

    service.CreateBarcodeCompleted += new EventHandler
        <BarcodeClient.BarcodeService.CreateBarcodeCompletedEventArgs>
        (service_CreateBarcodeCompleted);
    service.CreateBarcodeAsync(10, 10, 10, 10, "Title", 10, true,
        "one", "two", "three", "four", "five", 24, 8, true, "gif");
}

void service_CreateBarcodeCompleted(object sender,
    BarcodeClient.BarcodeService.CreateBarcodeCompletedEventArgs e)
{
    byte[] buffer = e.Result;
    MemoryStream stream = new MemoryStream(buffer);
    Bitmap bmp = new Bitmap(stream);
    pictureBox.Width = bmp.Width;
    pictureBox.Height = bmp.Height;
    pictureBox.Image = bmp;
}
```

Testing Web Methods in a Browser

- Using HTTP-GET

```
http://www.contoso.com/math.asmx/Subtract?num1=10&num2=5
```

- Using HTTP-POST

```
<form method="POST"
      action='http://www.contoso.com/math.asmx/Subtract'>
    <input type="text" size="5" name='num1'\"> -
    <input type="text" size="5" name='num2'\"> =
    <input type="submit" value="Subtract">
</form>
```

Exceptions in XML Web Services

SOAP Faults

- Exceptions thrown by a Web service method created using ASP.NET are sent back to the client in the form of a SOAP fault
- A SOAP fault is a **`Fault`** XML element within a SOAP message that specifies when an error occurred
  - It may contain details such as the exception string and the source of the exception
- Fortunately, both clients and Web services created using ASP.NET do not populate or parse the **`Fault`** XML element directly
  - The common design pattern for throwing and catching exceptions in the .NET Framework can be used

SoapException

- A Web service can throw either a generic **`SoapException`** or an exception specific to the problem
  - ASP.NET serializes the exception into a valid SOAP message by placing the exception into a SOAP **`Fault`** element
  - When the SOAP message is deserialized by a client, the SOAP fault is converted to a **`SoapException`** exception
    - The exception details are placed in the **`Message`** property
- When unhandled exception occurs while executing the Web service method, the exception is caught by ASP.NET and thrown back to the client
  - A .NET Framework client receives a **`SoapException`** with the specific exception placed in the **`InnerException`** property

SoapHeaderException

- A **`SoapHeaderException`** is thrown when an exception case was detected while processing a SOAP **`Header`** element
  - This exception is translated into a **`Fault`** element placed inside the response's **`Header`** element
  - A .NET Framework client receives the **`SoapHeaderException`**

SOAP Headers

Defining SOAP Headers

```
public class MyHeader : SoapHeader {
    public string Login;
    public string Password;
}

[WebService(Namespace = "http://www.contoso.com")]
public class MyWebService {
    // Add a member variable of the type deriving from SoapHeader.
    public MyHeader myHeaderMemberVariable;

    // Apply a SoapHeader attribute.
    [WebMethod]
    [SoapHeader("myHeaderMemberVariable")]
    public string MyWebMethod() {
        // Process the SoapHeader.
        if (myHeaderMemberVariable != null) {
            return String.Format("Login: {0}, Password: {1}",
                                 myHeaderMemberVariable.Login,
                                 myHeaderMemberVariable.Password);
        } else {
            return "-- no MyHeader --";
        }
    }
}
```

Building a Client That Processes Headers

```
class Program
{
    static void Main(string[] args)
    {
        Service.MyHeader myHeader = new Service.MyHeader();
        myHeader.Login = "John";
        myHeader.Password = "Kovalsky";

        Service.MyWebService srv = new Service.MyWebService();
        srv.MyHeaderValue = myHeader;
        string s = srv.MyWebMethod();

        Console.Write(s);
    }
}
```

Changing a SOAP Header's Recipients

```
public class MyHeader : SoapHeader
{
    public string Username;
    public string Password;
}

[WebService(Namespace = "http://www.contoso.com")]
public class MyWebService : WebService
{
    public MyHeader myOutHeader;

    [WebMethod]
    [SoapHeader("myOutHeader",
                Direction = SoapHeaderDirection.Out)]
    public void MyOutHeaderMethod()
    {
        // Return the client's authenticated name.
        myOutHeader.Username = User.Identity.Name;
    }
}
```

Handling Unknown SOAP Headers

```
public class MyWebService
{
    public SoapUnknownHeader[] unknownHeaders;

    [WebMethod]
    [SoapHeader("unknownHeaders")]
    public string MyWebMethod()
    {
        foreach (SoapUnknownHeader header in unknownHeaders) {
            // Check to see if this is a known header.
            if (header.Element.Name == "MyKnownHeader") {
                header.DidUnderstand = true;
            } else {
                // For those headers that cannot be
                // processed, set DidUnderstand to false.
                header.DidUnderstand = false;
            }
        }
    }
}
```

SOAP Extensions

SOAP Extensions

- SOAP extensions allow developers to augment the functionality of a Web service
  - For instance, an encryption or compression algorithm can be implemented to run with an existing Web service
- Typically, when a SOAP extension modifies the contents of a SOAP message, the modifications must be done on both the client and the server
  - For instance, if the SOAP message is not decrypted, then the ASP.NET infrastructure cannot deserialize the SOAP message into an object

Extending the SoapExtension Class

- The **`ChainStream()`** method is passed a **`Stream`** object and returns a **`Stream`** object
  - A SOAP extension should read from the **`Stream`** passed into **`ChainStream()`** and write to the **`Stream`** returned from **`ChainStream()`**
  - Therefore, it is important within the **`ChainStream()`** method to assign both **`Stream`** references to member variables
- The **`GetInitializer()`** and **`Initialize()`** methods are used to initialize internal data, based on the Web service or Web service method it is applied to
- Actual extended processing beyond the standard SOAP processing is performed by the **`ProcessMessage()`** method

Implementing the SOAP Extension

- There are two ways to run a SOAP extension on either a client or server application:
  - Configuring the application to run the extension (it can be done for all Web methods or all Web services)

```
<configuration>
    <system.web>
        <webServices>
            <soapExtensionTypes>
                <add type="Contoso.MySoapExtension, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=31bf3856ad364e35"
                     priority="1"
                     group="0"/>
            </soapExtensionTypes>
        </webServices>
    </system.web>
</configuration>
```

  - Creating a custom attribute that is applied to a Web service method

```
public class TraceExtension : SoapExtension {
    Stream oldStream, newStream;
    string filename;

    public override Stream ChainStream(Stream stream){
        oldStream = stream;
        newStream = new MemoryStream();
        return newStream;
    }

    // When the SOAP extension is accessed for the first time,
    // the XML Web service method it is applied to is accessed to
    // store the file name passed in, using the corresponding
    // SoapExtensionAttribute.
    public override object GetInitializer(LogicalMethodInfo  methodInfo,
                                        SoapExtensionAttribute attribute) {
        return ((TraceExtensionAttribute) attribute).Filename;
    }

    // The SOAP extension was configured to run using a
    // configuration file instead of an attribute applied to a
    // specific Web service method.
    public override object GetInitializer(Type WebServiceType) {
        return "C:\\" + WebServiceType.FullName + ".log";
    }

// ...
// ...
    // Receive the file name stored by GetInitializer and store it
    // in a member variable for this specific instance.
    public override void Initialize(object initializer) {
        filename = (string) initializer;
    }

    // If the SoapMessageStage is such that the SoapRequest or
    // SoapResponse is still in the SOAP format to be sent or
    // received, save it out to a file.
    public override void ProcessMessage(SoapMessage message) {
        switch (message.Stage) {
            case SoapMessageStage.BeforeSerialize:
                break;
            case SoapMessageStage.AfterSerialize:
                WriteOutput(message);        // custom method
                break;
            case SoapMessageStage.BeforeDeserialize:
                WriteInput(message);         // custom method
                break;
            case SoapMessageStage.AfterDeserialize:
                break;
            default:
                throw new Exception("invalid stage");
        }
    }
//...
```

```
// ...
    // Create a SoapExtensionAttribute for the SOAP Extension that
    // can be applied to a Web service method.
    [AttributeUsage(AttributeTargets.Method)]
    public class TraceExtensionAttribute : SoapExtensionAttribute {

        private string filename = "c:\\log.txt";
        private int priority;

        public override Type ExtensionType {
            get { return typeof(TraceExtension); }
        }

        public override int Priority {
            get { return priority; }
            set { priority = value; }
        }

        public string Filename {
            get { return filename; }
            set { filename = value;
        }
      }
   }
}
```