INTRODUCTION TO WEB SERVICES

What is a Web Service ?

Web service is a means by which computers talk to each other over the web using HTTP and other universally supported protocols.

A Web service is an application that:

- Runs on a Web server
- Exposes Web methods to interested callers
- Listens for HTTP requests representing commands to invoke Web methods
- Executes Web methods and returns the results

Web Services is based on:

- HTTP (Hypertext Transport Protocol)
- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WS-POLICY (Web Services Policy)

Most Web services expect their Web methods to be invoked using

HTTP requests containing SOAP messages. SOAP is an XML-based

vocabulary for performing remote procedure calls using HTTP and

other protocols.

Sample web service

Calc.asmx

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/WebService.cs" Class="WebService"
    %>
using System;
using System.Web.Services;

[WebService (Name="Calculator Web Service",
Description = "Perform simple math over the Web")]
class CalcService
{
[WebMethod (Description = "Computes the sum of two integers")]
public int Add (int a, int b) { return a+b;}
[WebMethod (Description = "Computes the difference between two integers")]
public int Subtract (int a, int b) { return a-b;}
}
```

The example demonstrates several important principles of Web service programming using the .NET Framework:

- Web services are implemented in ASMX files. ASMX is a special file name extension registered to ASP.NET (specifically, to an ASP.NET HTTP handler) in Machine.config.

- ASMX files begin with @ WebService directives. At a minimum, the directive must contain a Class attribute identifying the class that makes up the Web service.

- Web service classes can be attributed with optional WebService attributes. The one in the previous example assigns the Web service a name and a description that show up in the HTML page generated when a user calls up Calc.asmx in his or her browser.

- Web methods are declared by tagging public methods in the Web service class with WebMethod attributes. You can build helper methods into a Web service—methods that are used internally by Web methods but that are not exposed as Web methods themselves—by omitting the Webmethod attribute.

Testing a Web Service :

How do you test an ASMX Web service? Simple: just call it up in your browser.

ASP.NET responds to the HTTP request for Calc.asmx by generating an HTML page that describes the Web service.

- The name and description in the ASMX file's WebService attribute appear at the top of the page.
- Underneath is a list of Web methods that the service exposes, complete with the descriptions spelled out in the WebMethod attributes.

Click "Add" near the top of the page, and ASP.NET displays a page that you can use to test the Add method .

- ASP.NET knows the method name and signature because it reads them from the metadata in the DLL it compiled from Calc.asmx. It even generates an HTML form that you can use to call the Add method with your choice of inputs.
- The XML returned by the Web method appears in a separate browser window

The forms that ASP.NET generates on the fly from ASMX files enable you to test the Web services that you write without writing special clients to test them with.

Suppose you write a Web service that publishes Web methods named Add and Subtract that callers can use to add and subtract simple integers. If the service's URL is www.wintellect.com/calc.asmx, here's how a client would invoke the Add method by transmitting a SOAP envelope in an HTTP request. This example adds 2 and 2:

```
POST /calc.asmx HTTP/1.1
Host: www.wintellect.com
Content-Type: text/xml; charset=utf-8
Content-Length: 338
SOAPAction: http://tempuri.org/Add

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema instance"
    xmlns: xsd=http://www.w3.org/2001/XMLSchema
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <Add xmlns="http://tempuri.org/">
            <a>2</a>
     <b>2</b>
        </Add>
    </soap:Body>
</soap:Envelope>
```

And here’s how the Web service would respond:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 353

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
     xmlns:xsd=http://www.w3.org/2001/XMLSchema
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
   <AddResponse xmlns="http://tempuri.org/">
          <AddResult>4</AddResult>
          </AddResponse>
    </soap:Body>
</soap:Envelope>
```

The Web service’s job is to parse the SOAP envelope containing the inputs, add 2 and 2, formulate a SOAP envelope containing the sum of 2 and 2, and return it to the client in the body of the HTTP response. This, at the most elemental level, is what Web services are all about.

Web services written with the .NET Framework also allow their Web methods to be invoked using ordinary HTTP GET and POST commands. The following GET command adds 2 and 2 by invoking the Web service’s Add method:

```
GET /calc.asmx/Add?a=2&b=2 HTTP/1.1
Host: www.wintellect.com
```

The Web service responds as follows:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 80

<?xml version="1.0" encoding="utf-8"?>
 <int xmlns="http://tempuri.org/">4</int>
```

Here’s a POST command that adds 2 and 2:

```
POST /calc.asmx/Add HTTP/1.1
Host: www.wintellect.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

a=2&b=2
```

And here’s the Web service’s response:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 80

<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://tempuri.org/">4</int>
```

The hard part of writing a Web service is parsing HTTP requests and generating HTTP responses. The .NET Framework insulates developers from the low-level details of HTTP, SOAP, and XML and provides a high-level framework for writing Web services and Web service clients alike.

Web Services Description Language - WSDL

If other developers are to consume (that is, write clients for) a Web service that you author, they need to know :

- What Web methods your service publishes
- What protocols it supports
- The signatures of its methods
- The Web service’s location (URL)

All this information and more can be expressed in a language called the Web Services Description Language, or WSDL for short.

WSDL is an XML vocabulary http://www.w3.org/TR/wsdl.

Web Service Discovery—DISCO and UDDI

Once a client has a WSDL contract describing a Web service, it has all the information it needs to make calls to that Web service.

But when you publish a Web service by making it available on a Web server, how do clients find out where to get a WSDL contract? For that matter, how do clients know that your Web service exists in the first place?

The answer comes in two parts:

DISCO and Universal Description, Discovery, and Integration (UDDI)

- DISCO is a file-based mechanism for local Web service discovery—that is, for getting a list of available Web services from DISCO files deployed on Web servers.
- UDDI is a global Web service directory that is itself implemented as a Web service.