Master Project Defense

XML Transformer Web Service
XML-SQL Web Application

Defense Date: June 19, 2003

By

Sami I. Sarhan

Advisor
Dr. Gregory Riccardi

Committee Members

Dr. Lois Wright Hawkes
Dr. Robert A. van Engelen

Computer Science Department
Florida State University

Table of Content

1.0 Interface ………………………………………………………… 2
2.0 Introduction ……………………………………………………….. 3
3.0 XML Web Services ……………………………………………….. 4
3.1 Why XML Web Services? ………………………………………... 4
3.2 Standards and Protocols ………………………………………….. 5
3.2.1 SOAP ……………………………………………………………... 5
3.2.2 WSDL …………………………………………………………… 8
3.2.3 UDDI ……………………………………………………………... 8
3.2.4 XML …………………………………………………………… 9
4.0 Development XML Web Services in .NET Environment ………….. 10
4.1 ASP.NET ……………………………………………………… 10
4.1.1 ASP.NET vs. Traditional ASP ……………………………….. 10
4.1.2 ASP.NET and XML ………………………………………….... 10
4.2 Creating XML Web Service using VS .NET …………….……..11
4.2.1 Building the Add XML Web Service …………………….………12
4.2.2 Building a Web Client for the Add XML Web Service …….……..16
5.0 XML Transformer Web Service ………………………….……..19
5.1 XSL Transformations of XML ……………………….…………20
5.2 Building the Service …………………………………….……...21
5.3 Building Web Client ……………………………………….…..25
5.4 Building Console Client …………………………….…………29
5.5 Testing and Running …………………………….……………..30
5.5.1 Testing and Running the Web Client ……………………………32
5.5.2 Testing and Running the Console Client …………………….......34
6.0 XML-SQL Application …………………………………………34
6.1 Building Web Based Application ……………………………………36
6.2 Testing and Running ……………………………………………42
7.0 Conclusion ……………………………………………………44
8.0 References ……………………………………………...……. 45
Appendix A: Files Used and Descriptions ……………………..………46
Appendix B: XML Classes in .NET ……………………….……….…48
Appendix C: SQL Classes in .NET ………………………..……….…50
Appendix D: Source Code Attachments ………………..……….……51

1.0 Interface

The goals of this master project is to be able to achieve proficiency in creating XML Web Service in .NET Environment as well as develop skills in building XML-SQL Web Application using ASP.NET.

To achieve these goals I have set three main objectives: First, is to develop an XML Transformer Web Service, which will transform an XML document to another XML document given a stylesheet in this case it is an XSLT document. The two input parameters to the service will be the XML document, that we want to transform, and the XSLT document, which could be in different format (File, URL and Memory Stream). The service will return a string that contains the XML document.

Once we have built the service we would like to communicate with it. The second objective was to develop a consumer to the XML Transformer Web Service. The consumer will both be a web based and console based clients that will work as interfaces to the service.

The third objective is to develop XML-SQL Web Application. Given an XML document with SQL query and SQL connection string elements as an input parameter to the application. According to the connection string and the SQL query, the application will connects to the database and run the query. The result from the database will replaces the query string in XML format.

To meet the objectives I have gone through many activities some of which learning how to work with Visual Studio .NET (VS.NET) as well as learning how to build Web form in ASP.NET, which included examining different types of Web form components. This activity also included learning C# pronounced as "C sharp" .NET language. Knowing how to program in C and C++ made the learning of C# .NET easy and fast process.

In particular, XmlReader class and XmlWriter class was the focal point since I have used these classes extensively in this project. I have conducted other activities such as reading books and articles about developing XML Web Services in .NET environment. I have bought all the books that are listed in the reference section of this report. I read at least two chapters from each book and browsed the other chapters for reference.

According to the activities I have accomplished developing the XML Transformer Web Service and its both consumers the web based and the console based clients. Also, I have developed the XML-SQL application

2.0 Introduction

The Internet revolutionized how users interact with application specifically how computers talk to other computer by providing a universal data format that lets data be easily adapted or transferred. Before Web services, Internet computing and e-commerce were based on the exchange of information through enterprise application integration (EAI). Developers created one-time, proprietary solutions for system integration. Web services have emerged as the next generation of Web-based technology for exchanging information

Open standards and the focus on communication and collaboration among people and applications have created an environment where XML Web Services are becoming a platform for application integration. Applications are constructed using multiple XML Web Services from various sources that work together regardless of where they reside or how they were implemented.

The following sections are the main sections of this report. Starting at section three which will cover the building blocks for XML Web Service. Section four will cover what it takes to developing XML Web Service in .NET environment. We will build an Add XML Web Service and its consumer as an example for building XML Web Service with VS.NET. Section five will talk about the XML Transformer service and it consumers. Then in section six we will discusses the XML-SQL Web application development.

3.0 XML Web Services

"XML Web Services are a category of software components that provide functionality over the network" [3] this is very general definition to Web Services. However, if we would like to define Web Services in terms of functionality then XML Web Services let applications share data, and-more powerfully-invoke capabilities from other application without regards to how those applications were built, what operating system or platform they run on, and what device are used to access them. While XML Web Services remain independent of each other, they can loosely link themselves into a collaborating group that performs a particular task. [4]

3.1 Why XML Web Services?

There are many benefits to XML Web Services such as standard based, vendor neutral, simplicity, discoverable, and reduced development time. However, to answer the above question, in addition to the listed benefits, as a computer since student one main benefit of XML Web Services; in my opinion, they are language and platform independence. The only requirement for consuming an XML Web Services is the ability to communicate over TCP/IP and the ability to process XML.

XML Web Services do not enforce the use of any particular programming language or operating system. "A program written in C (a procedural language) running on handheld devices can consume an XML Web Service written in C# (an object-oriented language) that's running on Windows 2000 server" [3]

In addition to the language and platform independence, XML Web Services use industry standard protocols. Being standard based means that all XML Web Services implementations operate in the same way, use the same protocol and encode data consistency, which makes consuming XML Web Services a simple process no matter what platform the service and the consumer are running on.

3.2 Standards and Protocols

XML Web Services are invoked by means of industry-standard protocols including SOAP, WSDL, UDDI, and XML. They are defined through public standard organizations such as the World Wide Web Consortium (W3C) in the following sections we will discusses each of these underling technologies which are the building blocks of any XML Web Services.

3.2.1 SOAP

The Simple Object Access Protocol is the communication protocol for XML Web Services, which provides a standard way of packaging messages. A SOAP message is composed of an envelope that contains the body of the message and any header information used to describe the message, which is an XML document that follows specific XML schema. The latest version of SOAP is 1.1. However, "On July 9, 2001, a working group draft of SOAP 1.2 was published (http://www.w3.org/TR/2001/WD-soap12-20010709) by the XML Protocol Working Group" [2]

The major key aspects of the SOAP specification are as follows:

- **The SOAP envelope**: This is used to encode the header information about the message and the body of the message itself. Elements that can be included in to the header: Authentication, Security digits information, Routing information, Transactions, and payment information. The header entries appear as child nodes within the SOAP Header element. Here is an example:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Digest>B839D234A3F87</Digest>
  </soap:Header>
  <soap:Body>
    <StockReport>
      <Symbol>MSFT</Symbol>
      <Price>74.56</Price>
    </StockReport>
  </soap:Body>
</soap:Envelope>
```

- **SOAP encoding**: SOAP Encoding defines the way data can be serialized within a SOAP message. It builds on the types defined in the XML specification, which defines a standard way of encoding data within an XML document. Also, it clarifies how data should be encoded and covers items not explicitly covered in the XML specification, such as arrays and how to properly encode references.
- **RPC-style messages**: This is the protocol that facilitates procedure-oriented communication via request and response messages patterns. The SOAP 1.1 specification

described the recommended way to encode the request/response messages [2]. An example[1] of a request messages written in C#:

```
public int Add(int x, int y)
{ return x + y;}
```

Is as follows:

```
POST /project/Add/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://localhost/project/Add/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://localhost/project/Add/">
      <x>int</x>
      <y>int</y>
    </Add>
  </soap:Body>
</soap:Envelope>
```

The <soap:Body> element contains an <Add> element. Each of the input parameters is represented as a sub element (<x>, <y>) within the <Add> element. The order of the <x> and <y> elements must match the order in which the parameters are specified in the method.

An example of a response message that is used to communicate the results of the above SOAP request is as follows:

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

<?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://localhost/project/Add/">
      <AddResult>int</AddResult>
    </AddResponse>
  </soap:Body>
```

[1] Note: The SOAP, HTTP GET and HTT POST examples correspond to the actual service I have implemented for this report. The service will be discussed in detail in section 4.2.

```
</soap:Envelope>
```

? **The HTTP POST protocol binding**: This is the standard method of binding SOAP messages. Since HTTP protocol are supported by almost all current operating systems it became the typical way to send the message to a remote application. Some of the advantages of using HTTP protocol are:

- Firewall friendly: most firewalls have port 80, which are most of the time are open for HTTP traffic.
- Strong Support: many technologies have been introduced in the effort to increase the scalability and availability of HTTP-based applications.
- Stateless: the stateless nature of HTTP helps ensure that communication between the client and the server is reliable, especially across the Internet.
- Simple: the HTTP protocol is composed of a header section and a body section.
- RPC-style message exchanges: HTTP is a natural protocol for RPC-style communication because a request is always accompanied by a response.
- Open: practically every network-aware system supports HTTP.

Example HTTP GET of the Add Service is as follows:

```
GET /project/Add/Service1.asmx/Add?x=string&y=string HTTP/1.1
Host: localhost

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

<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://localhost/project/Add/">int</int>
```

Example HTTP POST of the Add Service is as follows:

```
POST /project/Add/Service1.asmx/Add HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

x=string&y=string

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

<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://localhost/project/Add/">int</int>
```

3.2.2 WSDL

Web Service Description Language (WSDL) is an XML-based language layered on top of the schema that describes a Web Service. It provides the information necessary for a client to interact with the Web Service. "WSDL specifies what a request message must contain and what the response message will look like in unambiguous notation."[5] WSDL is extensible and can be used to describe practically any network service, including SOAP over HTTP and even protocols that are not XML-based, such as DCOM over UDP.

The WSDL document contains five main elements under the <definition> elements that can be divided further in to two main sections. The first three elements are the abstract sections, which define SOAP messages in a platform and language independent manner; they do not contain any machine nor language specific elements [6]. They are:

1. **<types>**: contains schema definitions for the data exchanged between the client and the server. The default schema language is XML Schema. However, we can specify another schema language through the use of extensibility elements.
2. **<message>**: identifies a particular message that is exchanged between the client and the server. A message is composed of one or more parts. Each part is represented by the part element and can refer to an element or type definition defined within the <types> element.
3. **<portTypes>**: element contains one or more operation elements. We can think of an operation as an interface, a contract about how the client and the server will interact with each other to perform an action. An operation can be one of four types: request-response, solicit-response, one-way, or notification [2].

The remaining two elements are Site-specific matters such as serialization, which contain concrete descriptions. They are:

4. **<binding>**: is used to associate a port type with a particular protocol. This is accomplished via extensibility elements. Extensibility elements are elements defined outside the WSDL namespace. The WSDL specification defines three sets of extensibility elements for specifying binding information: SOAP, HTTP GET/POST, and MIME. Because specific technologies such as SOAP and HTTP are represented by extensibility elements, WSDL can be used to describe practically any service.
5. **<service>**: contains one or more port elements. A port element is used to define an address where a Web Service that supports a particular binding can be reached.

3.2.3 UDDI

Universal Description, Discovery, and Integration provide a central directory service for publishing technical information about Web Services. In other words, it is the yellow pages of Web Services. UDDI is the result of an industry initiative backed by a significant number of technology companies, including Microsoft, IBM, SAP, and Ariba. (You can find a full list of participants in the UDDI project at http://www.uddi.org/community.html.)

The infrastructure that supports UDDI is composed of a set of registries and registrars. A registry contains a full copy of the UDDI directory; a registrar provides UDDI registration services on behalf of a customer. A business must choose a registry in which to maintain its information. All updates made to the directory will be replicated to all the other registries. Then the updated information can be queried from any registry.

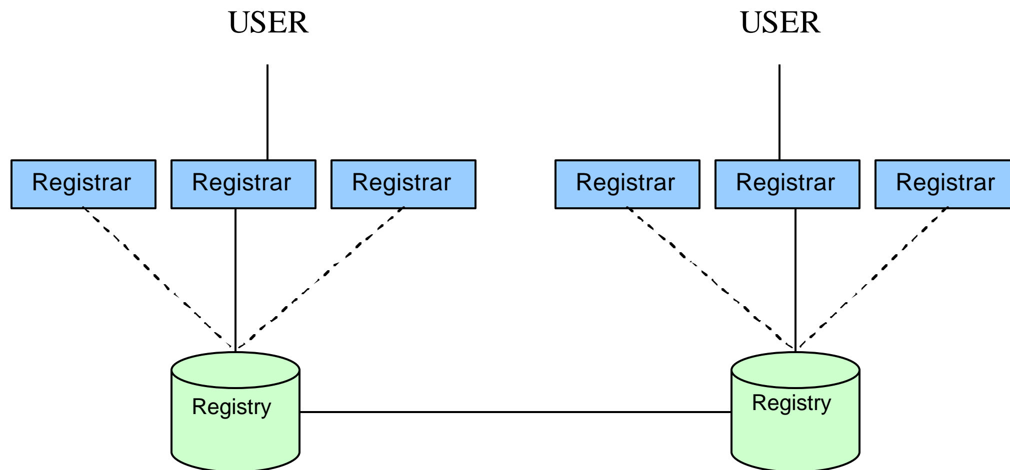

Figure 1

Figure 1 shows one user updating the UDDI business directory through a registrar and then another user accessing the updated information.

Microsoft introduced new technology that facilitates the publishing and discovering of XML Web Services this is the Discovery (DISCO) files. DISCO files are XML documents with an extension .disco. They include: URL reference to WSDL and URL reference to other DISCO files.

3.2.4 XML

All the above protocols and standards we discussed so far are based on the Extensible Markup Language. Indeed, XML is the backbone of any Web Service. "XML is a vehicle for information that brings usable data to the desktop and is a universal data format that does for data what HTML does for web content- it provides the necessary markup"[7]

The extensibility of XML language what makes XML the first choice for programmer to create richly structured documents that could be used in the Web. Compared to HTML, in XML we can add new tags and new elements to support our application. Whereas in HTML we are limited to the tags provided by the language. Also, HTML comes bound with a set of semantics and does not provide arbitrary structure. However, XML is a Meta language for describing markup languages and provides a facility for defining tags and structural relation between them.

4.0 Development XML Web Services in .NET Environment

Microsoft .NET allow programmer to develop applications in language independent manner. It has been specifically designed with Web Services in minds. VS.Net is the IDE (Integrated Development Environment) for .NET based application. I am using the academic version of VS.NET, which contains the .NET Framework SDK (Software Development Kits) and the main .NET languages VB (Visual Basic), C++, C# and Java script (Jscript) .NET.

All the XML capabilities that where formally available to the programmer through the Microsoft XML (MSXML) parsers are now encapsulated in the system.Xml namespace of classes. Refer to Appendix B for the list of classes we used in this project. In this project C# was the main language used to develop the XML Web Services along together with ASP.NET Web forms. In the following subsections we will talk about ASP.NET and building Web Services and its consumers in .NET

4.1 ASP.NET

The key to .NET technology for developing XML Web Services is ASP.NET, which is the next generation of ASP (Active Server Pages) platform. ASP.NET provides framework for developing Web based applications in which the user interacts with a service using a browser. Although the main emphasis of ASP.NET is to build conventional web applications, Microsoft has added support for building XML Web Services as well.

4.1.1 ASP.NET vs. Traditional ASP

ASP.NET provides a true language neutral execution framework for web application to use. This means that whether we use C#, VB, Jscript .NET, C++ or Perl our code will compile to IL (Intermediate Languages) and then executed by .NET Framework. This also means that ASP.NET will take full advantages of .NET Framework based classes through and compiled languages. However, compared to the traditional ASP, the choice of languages are limited with VB script and Java script.

Another major difference between traditional ASP and ASP.NET; is that traditional ASP pages are parsed each time they are requested, which will make the ASP execution slower. On the other hand, when we execute an ASP.NET page for the first time it gets compiled in to a binary .dll (dynamic library link) and then get executed. All further requests are served by this compiled code, making execution faster.

4.1.2 ASP.NET and XML

The relation between ASP.NET and XML can be summarized but not limited to the following points:

- ? ASP.NET uses XML extensively to represent Web Controls and configuration setting. It relies on special XML document for configuration purposes it is the web.config file refer to Appendix A for the description of this file.
- ? ASP.NET provides Web Controls that can deal with XML data and display it as our requirement, which can be either tabular or style sheet-based.
- ? The creation of XML Web Services in .NET platform is done through ASP.NET. We will see how we can create a Web Service in .NET and a client in the next section.

4.2 Creating XML Web Service using VS .NET

There is always two parts of any XML Web Service: the Web Service itself and the consumer of the service in this case the client of the Web Service. The client could be a Web application, Windows application, server process, and another XML Web Service. In this section I will illustrate how to build both an XML Web Service and a Web client to the service using VS .NET.

The example I will use is a simple Add XML Web Service that will add two numbers and return the result. But before I build the service and its consumer Figure 2 will illustrate for us the role of the proxy class, which we will discuss in detail in section 4.2.2 Building a Web Client for the Add XML Web Service.

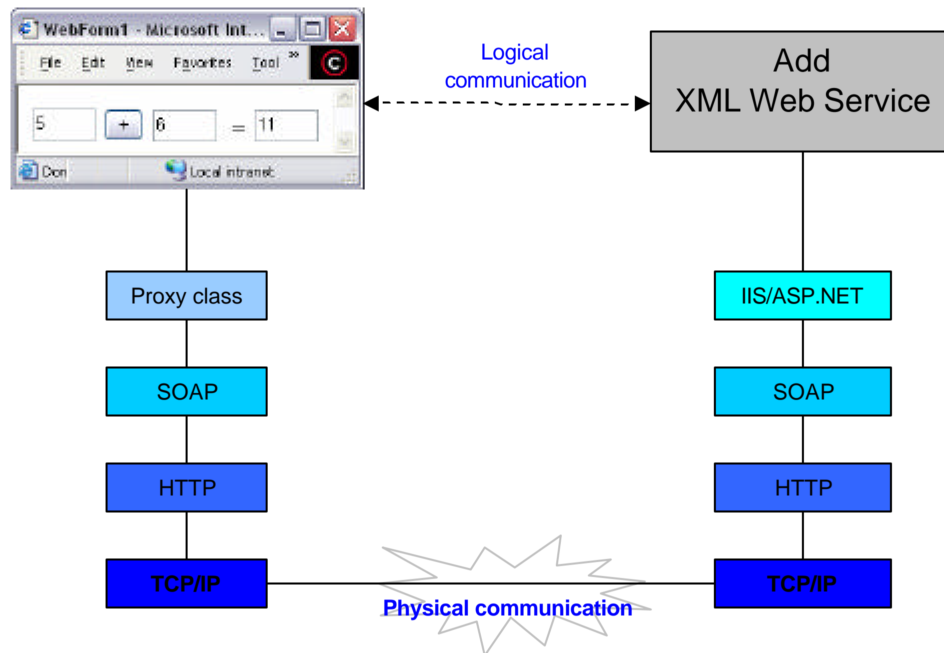

Figure 2

In the above figure the client application is unaware of the activates that are happening once the call is made to the proxy method. Since XML Web Service interfaces are defined using

WSDL, VS.NET can auto generate the proxy class for us, which will takes the complexity of SOAP processing out of the application code.

4.2.1 Building the Add XML Web Service

We will build simple addition Web Service given two numbers to the service as an input parameter the service will add them and return the result. We will implement what we discussed in earlier section when we talked about SOAP. When we chose to create new project in VS.NET one of the template option under the C# projects is ASP.NET Web Service as it is in Figure 3:

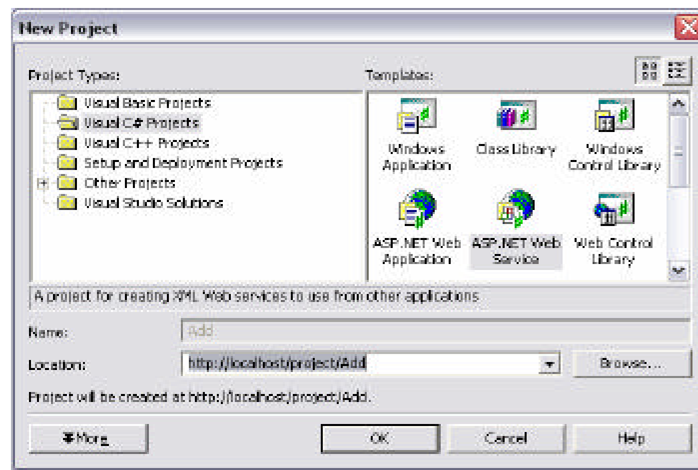

Figure 3

I assume that IIS (Internet Information Server) in installed. As soon as we click the OK button VS.NET start to communicate with IIS and create a new virtual directory for the new service under the specified location (http://localhost/project/Add).

VS.NET will create the service class and give a name Service1.asmx, which can be renamed. We add a new method and name it Add it is in the code below[2]:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace Add
{
[WebService(Namespace="http://localhost/project/Add/", Name="AddService",
Description="XML Web Service for adding two numbers")]
```

[2] Note: that I have removed some of the auto generated code by VS.NET. For the full code refer to Appendix E.

```
public class Service1 : System.Web.Services.WebService
{
	public Service1()
	{	//CODEGEN: This call is required by the ASP.NET Web //Services
		Designer
		InitializeComponent();}

	[WebMethod]
	public int Add(int x, int y)
	{ return x + y; }
}}
```

We can notice a few things from the code above:

1. The using of the System.Web namespace. It supplies classes and interfaces that enable browser-server communication. This namespace includes the HttpRequest class which provides extensive information about the current HTTP request, the HttpResponse class which manages HTTP output to the client, and the HttpServerUtility class which provides access to server-side utilities and processes. System.Web also includes classes for cookie manipulation, file transfer, exception information, and output cache control.
2. The System.Web.Services namespace consists of the classes that enable us to create XML Web Services using ASP.NET and XML Web Service clients. It also, defines the optional base class for XML Web Services, which provides direct access to common ASP.NET objects, such as application and session state.
3. The two attribute [WebService], which is optional, and [WebMethod], which is required.
    a. The [WebService] attribute, which is placed in top of the class name, is used to add additional information to an XML Web Service, such as a string describing its functionality and its name as I did in the above code.
    b. The [WebMethod], which must be placed on any method that we want to programmatically expose over the Web. Adding this attribute to a method within an XML Web Service created using ASP.NET makes the method callable from remote Web clients. This class cannot be inherited. The method and class must be public and running inside an ASP.NET Web application.

Now that we have created the Web Service, we can test it using the auto generated form/interface by the ASP.NET. By hitting F-5, the program will be compiled and run by opening Internet Explorer (IE) to the Web Service front page. As it is in Figure 4.

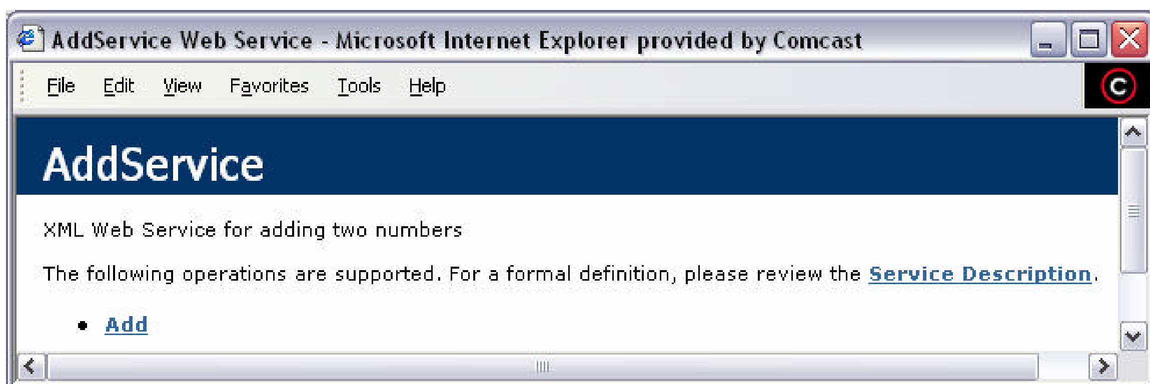

Figure 4

Notice that the name and the description we used in the [Web Service] attribute are shown in the front page of the service. Even though this page was auto generated by ASP.NET, we can customize it by adding extra information about the service or even pictures and background.

Also, we notice there are two links available for us to click one is the Add link and the other is the service description link. Let's examine the Add link first.

Before we activate this link, let's ask this question: Assume we had another method/s in our class will they show up in this front page as clickable link? The answer is NO. If we want to show another method that can be clicked and tested then we must add the [WebMethod] attribute in top of it. I will add two more methods Mult() and Sub() to the code. The Mult() method will not have the [WebMethod] attribute in top of it but the Sub() will have as it is below.

```
……

……
public int Mult(int x, int y)
{return x*y;}

[WebMethod]
public int Sub(int x, int y)
{return x - y;}
```

The expected front page for the service is in Figure 5.

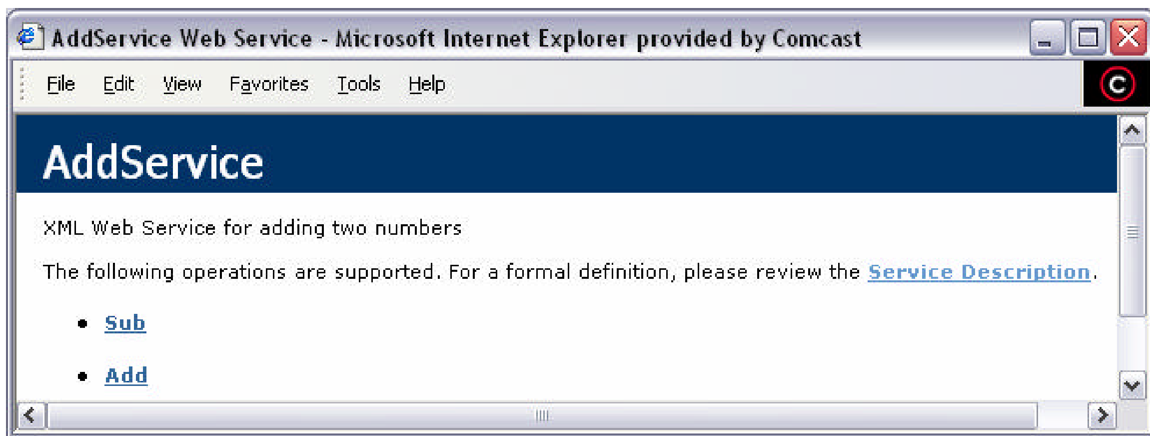

Figure 5

Also, notice that the WSDL file will change accordingly since now we have access to two methods instead of one through the service.

Back to our original discussion if we click the Add link we will get the ASP.NET form. We can enter the values we need to add in the form, then by clicking Invoke button we will get the result in XML document opened in new browser window. As it is shown in Figure 6 and 7 respectively.

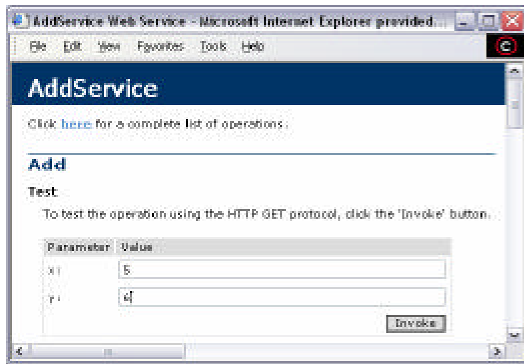

Figure 6

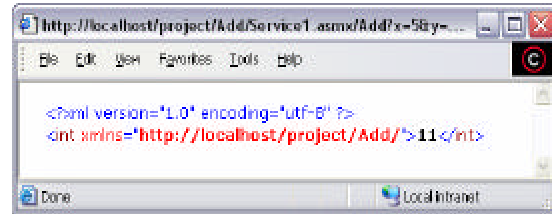

Figure 7

http://localhost/project/Add/Service1.asmx?WSDL - Microsoft Internet Explorer provid...

File Edit View Favorites Tools Help

```
  <?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://localhost/project/Add/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://localhost/project/Add/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  + <types>
  + <message name="AddSoapIn">
  + <message name="AddSoapOut">
  + <message name="AddHttpGetIn">
  + <message name="AddHttpGetOut">
  + <message name="AddHttpPostIn">
  + <message name="AddHttpPostOut">
  + <portType name="AddServiceSoap">
  + <portType name="AddServiceHttpGet">
  + <portType name="AddServiceHttpPost">
  + <binding name="AddServiceSoap" type="s0:AddServiceSoap">
  + <binding name="AddServiceHttpGet" type="s0:AddServiceHttpGet">
  + <binding name="AddServiceHttpPost" type="s0:AddServiceHttpPost">
  - <service name="AddService">
      <documentation>XML Web Service for adding two
        numbers</documentation>
    - <port name="AddServiceSoap" binding="s0:AddServiceSoap">
        <soap:address
          location="http://localhost/project/Add/Service1.asmx" />
      </port>
    - <port name="AddServiceHttpGet" binding="s0:AddServiceHttpGet">
        <http:address
          location="http://localhost/project/Add/Service1.asmx" />
      </port>
    - <port name="AddServiceHttpPost" binding="s0:AddServiceHttpPost">
        <http:address
          location="http://localhost/project/Add/Service1.asmx" />
      </port>
    </service>
  </definitions>
```

Figure 8

The other link that was available for us to click in Figure 4 is the service description link, which is the link to the WSDL file as it is in Figure 8.

We can see the five elements we have discussed earlier in the WSDL section and for the demonstration purpose I have collapsed all the nodes axcept the service node. In the <service> element, which specifies port address/es of each binding, as well as the http address location in this case the URL to the .asmx file, which is a .NET convention for XML Web Service files.

By now our service is ready we can consume this service and call it from another application. Next we will discuss the process of creating a Web client that will communicate to the Add XML Web Service and in particular we will elaborate on the proxy object in .NET framework. But before we move on to the next section it worth to mention that the page that included the ASP.NET auto generated form for our service also it included the sample request and response for SOAP, HTTP GET, and HTTP POST and those are the examples we used above in the SOAP section earlier.

4.2.2 Building a Web Client for the Add XML Web Service

In this section I will build a web based client for the Add XML Web Service we developed in the previous section and go through the steps of adding the service to the application before we used it.

XML Web Service consumers do not need to know the detail of the platform or the language used to implement the XML Web Service; the only requirement as I said earlier in section 3.0 is the ability to communicate over TCP/IP and the ability to process XML. What I mean by process, is to be able to formulate request and process response using correct protocol and message structures, which are defined in the WSDL of the service.

It is possible to manually encode complex data as SOAP messages; however, .NET introduced a proxy class with methods mirror the functionality exposed by an XML Web Service [3]. In other words, the proxy object is the glue between the .NET Framework and an XML Web Service. Each proxy method takes the same number and type of arguments and returns the same data types as its XML Web Service equivalent.

To call XML Web Service's functionality, a client application simply calls the proxy class method. Which will then takes care of all the communications with XML Web Service and returns the response it receives from the service to the client application. In this case the XML Web Service invocation appears to be a local method call, while in reality the call could be serviced by an XML Web Service anywhere in the Internet.

Now we will go over the steps for add any XML Web Services in to an application that is developed by VS.NET:

Step1: Create the Proxy Class

Although, it is possible to create proxy classes using the Wsdl.exe command line tools supplied with the .NET Framework SDK. However, I will walk through adding a Web Reference to the client using .NET wizards that will create the proxy class for us. After we choose the new

ASP.NET Web Application from the new project menu on the Solution Explorer highlight References and right click Add Web Reference. We will get the Figure 9.

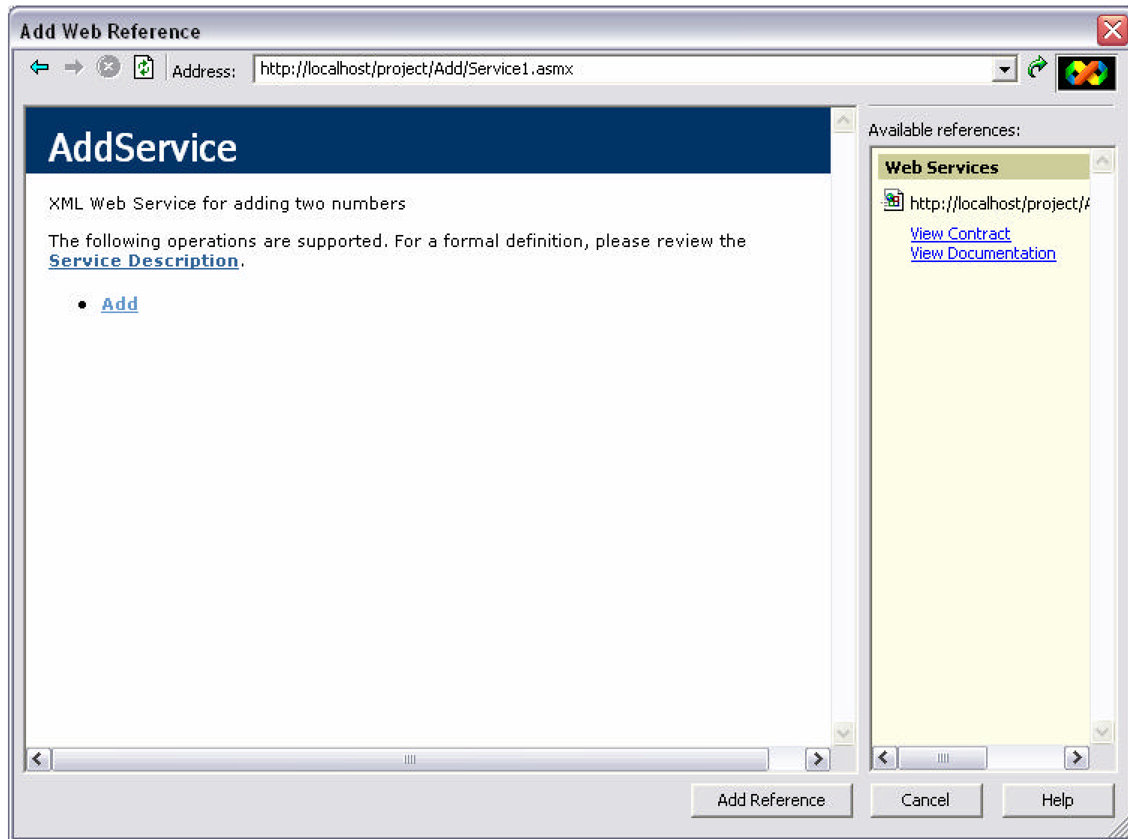

Figure 9

We need to type the URL for our XML Web Service. In this case the Service was located in my machine, but it could be located any where on the Internet. The URL for our Add Service is: http://localhost/project/Add/Service1.asmx as we said before the .asmx file extensions are .NET convention for XML Web Services. However, as well this could be a service that was created by non-Microsoft product. As long as the wizard is able to locate the WDSL file for the service to create the proxy class.

Once the service was added successfully that's mean proxy class was auto generated by VS.NET and placed under the Web Reference folder with a new directory that corresponds to the service name. For example, the path to the directory of the Add service is: C:\Inetpub\wwwroot\project\AddApp\Web_References\AddServ. Under the AddServ directory there exist four files that are used by the application refer to Appendix A for full detail of these files. The proxy class will always be given the name References.cs[3] and should not be edited manually.

[3] C# files ends with .cs

Step 2: Enabling the Proxy class

Once we are successful in adding our web reference we will enable the proxy class by first importing the namespace for the proxy class. That is done in C# by adding the following line of code:

```
using AddApp.AddServ;
```

AddApp corresponds to the project name and the AddServ is the name given by me when I added the web reference I had the choice of renaming it from localhost the given name by VS.NET to AddServ.

Second by instantiating the proxy class and calling the Add() method. That is done as well in C# by adding the following code:

```
AddService Number = new AddService();
```

Remember the AddService is our Service Name that we defined it in the [WebService] attributes and the Number is our proxy object. The proxy instance is stateless and thread safe; multiple objects and threads can use the proxy repeatedly. Now we can call the methods of the proxy class in this case we only have one method Add() and pass the right argument to it:

```
Result.Text = Number.Add(Numb1, Numb2).ToString();
```

Since our client is a web form, then we needed the ToString() method to convert the integer to a string to be able to post the result in to the Result text box. The following code corresponds to the client I avoided VS.NET auto generated code for simplicity:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

using AddApp.AddServ;

namespace AddApp
{
	public class WebForm1 : System.Web.UI.Page
	{
		protected System.Web.UI.WebControls.TextBox Num1;
```

```
            protected System.Web.UI.WebControls.Button Add;
            protected System.Web.UI.WebControls.TextBox Num2;
            protected System.Web.UI.WebControls.Label Equale;
            protected System.Web.UI.WebControls.TextBox Result;

            private void Add_Click(object sender, System.EventArgs e)
            {
            // Initialization
                  int Numb1 = 0;
                  int Numb2 = 0;

            // Parsing the String Text Box in to the Numb1 and Numb2
                  Numb1 = int.Parse(Num1.Text);
                  Numb2 = int.Parse(Num2.Text);

            //Instantiate the proxy class
                  AddService Number = new AddService();

            // Call the method of the proxy class and convert the //result in
            to String
                  Result.Text = Number.Add(Numb1, Numb2).ToString();

            }
      }
}
```

The interface for the form will look like Figure 10:

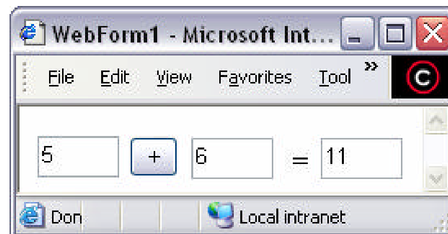

Figure 10

Now that we have built a good background on how to build an XML Web Services in .NET environment and build a client that will consume the service, in the next section I will discuss the XML Transformer Service and its client from an application point of view. In other words, I will be focusing on the particular XML methods in .NET and there capabilities of building XML based application.

5.0 XML Transformer Web Service

XML documents can contain all sorts of information that are some time not important to its consumers. To extract the useful data from an XML document and leave the others, there are two main tools that are available to programmers for displaying XML content in different format and extracting specific elements. One of these tools is the Cascading Style Sheet (CCS), which is the most familiar method for transforming XML documents for rendering on the Web. CSS allows

document authors to specify the presentation of elements on a Web page separately from the structure of the document. This separation of structure from presentation simplifies maintaining and modifying a document layout [8].

However, like any other technology there are drawbacks for using CSS to transform XML document for real-life application. For example, some of these drawbacks are CSS enables us to format and display XML elements only, where in any XML document transforming attributes is as important as transforming elements. CSS lacks the manipulation of records and displaying of data dynamically, it can only display static data.

In the next section we will discuses the second tool for transforming XML document which is the Extensible Stylesheet Language Transformation (XSLT). And we will see how we can overcome CSS drawbacks. Then we will talk about .NET support for XSLT.

5.1 XSL Transformations of XML

The goal of the XSLT is to transform the content of a source XML document into another document that is different in format or structure. For example, to transform XML into HTML for use on a Web site or to transform it into a document that contains only the fields required by an application. This transformation process is specified by the W3C XSL Transformations (XSLT) Version 1.0 recommendation located at http://www.w3.org/TR/xslt. XSLT is a technology that can perform tighter manipulation of XML than CSS [7].

Transforming an XML Document using XSLT to another XML document involves two tree structures. One is the source tree, which is the XML document that we want to transform. Two is the result tree, which is the created XML document. In the .NET Framework, the XslTransform class, found in the System.Xml.Xsl namespace, is the XSLT processor that implements the functionality of this specification. Figure 11 shows the transformation architecture of the .NET Framework [9].

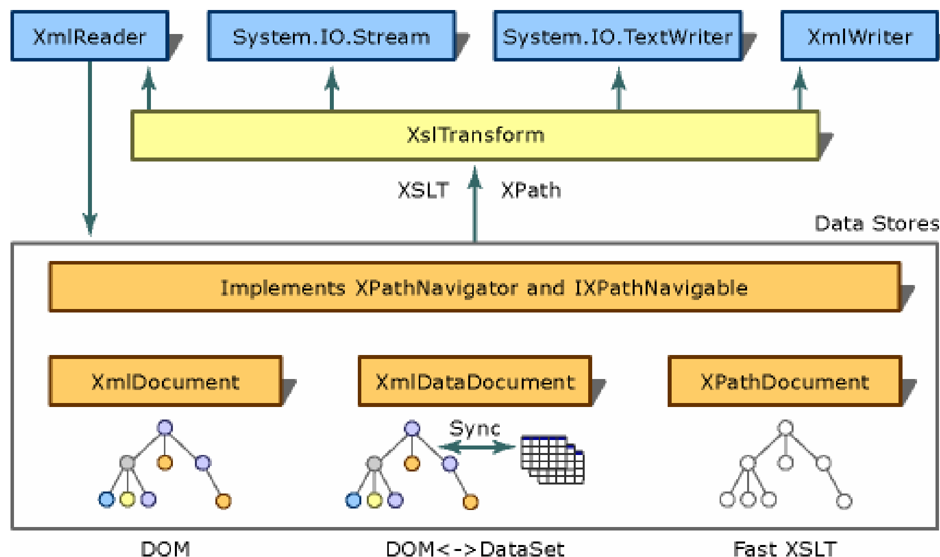

Figure 11

The XSLT recommendation uses XPath expression parser and evaluator, which are found in the System.Xml.XPath namespace, to select parts of an XML document. XPath is a locator language for XML document that enables us to select one or more nodes from an XML document that match the conditions in the locator. The .NET Framework implementation of XPath is used to select parts of an XmlDocument, an XmlDataDocument, and an XPathDocument as shown figure 11.

The XPathDocument class optimizes XSLT data store, and when used with XslTransform provides highly performed XSLT transformations. In the next section we will discus the programming point of view for building an XML transformer Web Service using .NET XML classes: XPathDocument, XslTransform, XmlReader, and XmlWriter.

5.2 Building the Service

We have talked about the building blocks of developing an XML Web Services in .NET in previous sections. In this section we will rather discuss the code needed to build our XML Transformer Service. I have divided this section in to four major steps for building the Web Service.

Step1: Reading the source documents using XmlReader class

Since we will not be using the direct interface/form that is auto generated by ASP.NET, we want to make sure that we can read both the XML and XSLT documents in all formats. By that I mean if the document is coming to us from Internet URL, file stream or string in the memory. The service should be able to handle any of these sources or a mixture of them. For example, the XSLT document could be residing in a local hard disk and the XML document can only be reached by a URL, then the service should be able to accept both inputs and process the transformation and return the result to the consumer of the service.

This step implemented by the ReadInputDocument() method. The method code as follows:

```
private XmlTextReader ReadInputDocument(string Doc)
{
	XmlTextReader DocReader;

	if (Doc.EndsWith("xml") || Doc.EndsWith("xsl"))
	{
		DocReader = new XmlTextReader(Doc);
	}
	else
	{
		// Memory stream reading the XML Document/Data
		byte[] byteDoc = Encoding.UTF8.GetBytes(Doc);
		MemoryStream DocStream = new MemoryStream(byteDoc);
		DocReader = new XmlTextReader(DocStream);
	}
	return DocReader;
```

```
        DocReader.Close();
}
```

As you can see from the above code we are passing one string value to this method string Doc, it could be the either the XML input document or the XSLT document.

In this method we are using one of the implementation for the XmlReader class which is the XmlTextReader. The XmlReader abstract class is a forward only reader class; we will cover XmlReader class in detail when we talk about the XML-SQL Web Application. In short, the XmlReader provides methods that can be used to navigate through the document node-by-node.

The XmlTextReader class is the most useful version of XmlReader. It enforces the rules that XML must be well formed. It is neither a validating nor a non-validating parser since it does not have DTD or schema information.

We start by declaring a new instance in XmlTextReader class:

```
XmlTextReader DocReader;
```

The XmlTextReader can read data from different inputs, such as a stream object, a TextReader Class object, and a URL identifying a local file location or web site.

The purpose of the if statement in the method is to be able to distinguish the type of the input string we are getting and add credibility to the service to be able to accept any form of input (File, URL, Memory Stream). The first branch of the if statement will resolves inputs strings that are either files located in the local hard disk or files located in the internet.

```
if (Doc.EndsWith("xml") || Doc.EndsWith("xsl"))
      {
            DocReader = new XmlTextReader(Doc);
      }
      else
      {
            // Memory stream reading the XML Document/Data
            byte[] byteDoc = Encoding.UTF8.GetBytes(Doc);
            MemoryStream DocStream = new MemoryStream(byteDoc);
            DocReader = new XmlTextReader(DocStream);
      }
```

The else branch of the if statement will load document from a string in memory. We are using the MemoryStream form of the constructor. And since MemoryStream only has byte array constructor, not a string constructor, we first need to move the string in to byte array then construct the MemoryStream object DocStream.

Finally, the method will return the XmlTextReader object DocReader and close the XmlTextReader object.

Step 2:Create XPathDocument Instance:

As we said earlier XPath is a query language used to navigate nodes of a document tree. We are using an instance of XPathDocument to hold the document being transformed. The XPathDocument is a performant cache, for processing documents with XslTransform. It is structurally similar to the XML DOM, but it is highly optimized for XSLT processing and the XPath data model using the XPath optimization functions on the XPathNavigator [10].

This is implemented in the [WebMethod] xmlToxml(). The following line of code establishes this step:

```
XPathDocument xmlDoc = new XPathDocument (ReadInputDocument(xmlinput));
```

The main method of the our service is the [WebMethod] xmlToxml(). This method will pass in two string values one for the XML document and the other is for the XSLT document. The string xmlinput will be passed to our reader method that will return the XmlTextReader object that XPathDocument will used it to navigate through the document.

Step 3: Creating XslTransform Instance:

XML documents are transformed in .NET framework using XslTransform class. This class constructor takes no parameters. The transformation is done in tree sub steps:

1. We create an instance of the XslTransform. This step will be established by the following line of code:

   ```
   XslTransform xslDoc = new XslTransform();
   ```

2. Then we need to use the Load() method to load the stylesheet to be used for the transforms. In this case the string xslinput will be passed to our reader method that will return the XmlTextReader object. This step will be established by the following line of code:

   ```
   xslDoc.Load(ReadInputDocument(xslinput));
   ```

3. Like the Load() method, the Transform() method has several overloads. In this case; however, it is important for us to carefully consider which form of the method we will use to perform the transformation. Ideally, we should use an instance of XPathDocument to hold the document being transformed. This step will be established by the following line of code:

   ```
   xslDoc.Transform(xmlDoc, null, myWriter);
   ```

In this case the three arguments of the Transform() method are: the XPathDocument object, XSLT argument list object in this case is null, and the XmlTextWriter object. Before the compiler executes this line of code it has to have the XmlTextWriter object created, which is the topic of the final step for building the transformer service. The XsltArgumentList class contains XSLT parameters and XSLT extension objects. When passed to the Transform these parameters and extension objects can be invoked from stylesheets.

Step 4: Create an XmlTextWriter to handle the output:

The final Step in building the XML Transformer Service is to create the XmlTextWriter object. Just like the XmlTextReader, XmlTextWriter, derived from the XmlWriter, writes XML to a file, console, stream, and other output types. Also, it represents a writer that provides a fast, non-cached, forward-only way of generating streams or files containing XML data that conforms to the W3C XML 1.0 and the namespaces in XML recommendations.

Some of the tasks are done by the XmlTextWriter to ensure well-formed XML are as follows:

? Ensures that the XML elements are written out in the correct order. For example, it will not let you write an attribute outside of an element, write a CDATA block inside an attribute, or write multiple root elements.

? Ensures that value and format of the *xml:space* attribute is correct and makes sure that its value is acceptable according to the XML 1.0 2nd recommendation (www.w3.org/XML/Group/2000/07/REC-xml-2e-review#sec-white-space) [11].

? Checks when a string is used as a parameter, for example *Null==String.Empty* and *String.Empty* and whether it follows the W3C rules.

The XmlTextWriter constructors creates an instance of the XmlTextWriter, which takes a filename, stream, or TextWriter. An overloaded method exists to take an additional parameter that defines the encoding type. The following line of code creates an instance of XmlTextWriter:

```
XmlTextWriter myWriter;
```

Since we are not sure how the consumer would like to store the transformed document we will return the document in a string format. In this case the XmlTextWriter will send the data to a MemoryStream. Then we can use the stream form of the constructor. The following line of code will establishes this task for us:

```
Stream s = new MemoryStream();
myWriter = new XmlTextWriter(s, null);
myWriter.Formatting = Formatting.Indented;
```

The Formatting properties will defines whether indenting is used to format the output.

Now our object is ready to be used. Back to step three above, we can now pass the XmlTextWriter object to the Transform() method. Once the transformation if done we would like

to read the memory and extract the transformed string then return it to the client/consumer. The following line of code will established this task:

```
myWriter.Flush();
s.Position = 0;
StreamReader sr = new StreamReader(s);
return sr.ReadToEnd();
```

In this case we used the StreamReader to read the MemoryStream. The Flush() method would clears all buffers for the current writer and causes any buffered data to be written to the underlying device.

By now our service is ready to be used. In the next section we will go over the process of creating the client that will consume this service.

5.3 Building Web Client

We have gone through building a simple client to use .NET created Web Service in previous sections. Since one of the goals of this project is to become an experienced ASP.NET programmer. In this section I will go though the steps of creating an ASP.NET web form for our service. But, before we jump in to the code we would like to ask the following questions:

- How the users will be interacting with the service? In other world we need to provide user friendly Web form.
- Also, after the user will get the transformed document, what they will do with it? In other words, are they going to view it, if yes, how? Will they save it to local hard disk?

To answer the first question I thought of a simple Web form that will contain user friendly components such as buttons, text boxes with label on each text box, and radio buttons that will allow the users to choose one option at time. Also, in case the user will make a mistake a reset button is provided to for re-entering the data.

To answer the second question, after the user will get the transformed document and feel that he/she wants to use it for other purpose such as saving the document in local hard disk. Or view it on the Internet Explorer (IE) which has the capability to view XML document in nice hierarchal view.

Now that we have in mind users expectation on how to interact with the service. I thought about having three main text boxes: one for the XML document, one for the XSLT document and the last one is for the result document. In ASP.NET these text boxes are set by default to accept single line. To allow the user to copy any of the input documents and pastes it to the text box we have to change the property of the text box to accept Multi line. For this reason, during the development of the ReadInputDocument() method, we discussed the in the previous section, we considered that the document could be coming as a string from the memory not just a URL to a file either in local disk or in the Internet.

After the user will get the result document he/she will have at least three choices of what to do with the transformed document. One option would be just to view it in IE browser. The other option is to save it in to local disk for later use or to a directory that will be used by another application. The last option I have is to send it a container that is a text box in a new Web page. Which is a regular behavior of many forms in the Internet where the result could be viewed in a different Web page.

From the above dissection we know that we need to have at least three buttons: one for the transformation, one for the reset, and the last one is for what to do with the result document. Figure 13 shows the final page that was developed in ASP.NET.

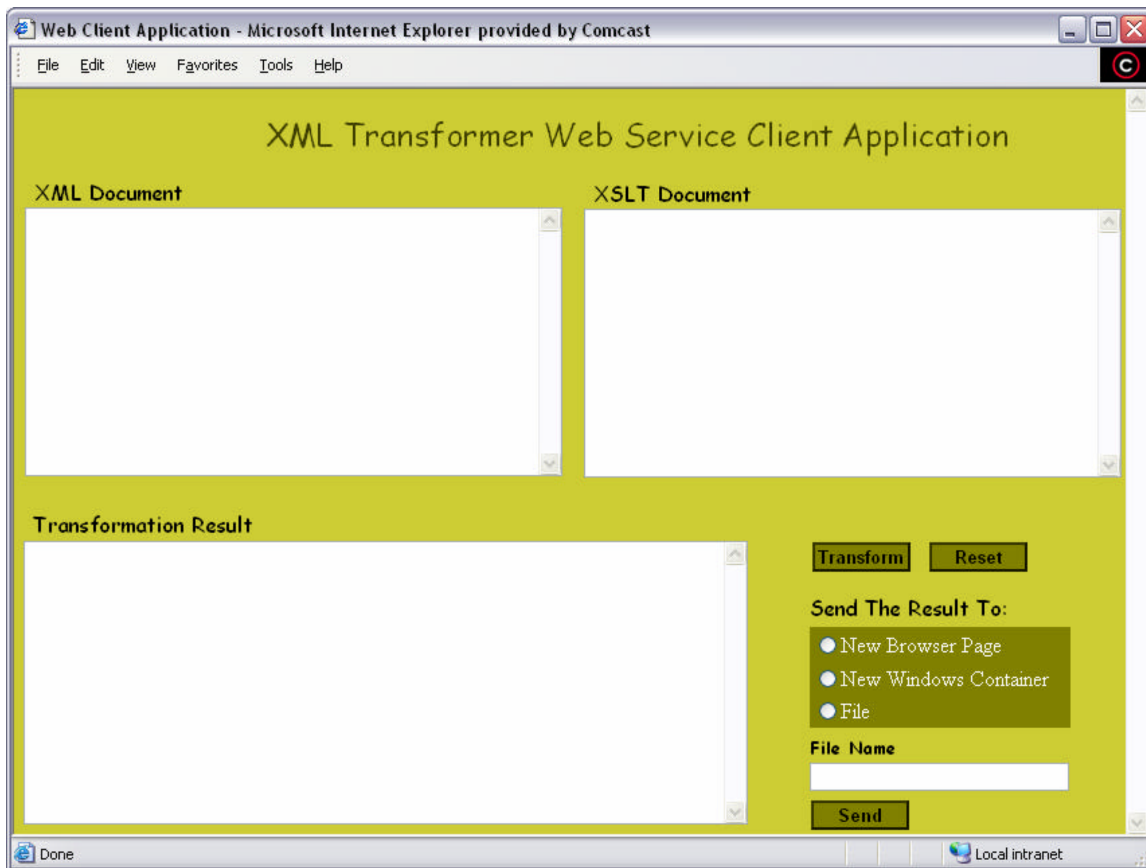

Figure 13

Now we can move forward to the code behind this form and look in to the action behind the major buttons. To understand the code Table 1 lists the components of the Web form along with there names.

Components Name	Components Type	Description
`txtXML`	Text Box	XML document
`txtXSL`	Text Box	XSLT document
`txtResult`	Text Box	Transformed/Result document

ResultFile	Text Box	Result document file name
Transform	Button	Execute transformation
Reset	Button	Clear all content
Send	Button	Send the result according to the radio button selection
File	Radio Button List value	Send the result to a file specified in the file name text box
New_Windows_Container	Radio Button List value	Send the result to new Web page container
New_Browser_Page	Radio Button List value	Open the result in new IE browser window

Table 1

We will start by looking in to the code behind the "Transform" button:

```
private void Transform_Click(object sender, System.EventArgs e)
{
	//Instantiate the proxy class
	Trans x_Trans = new Trans();

	// Call the method of the proxy class
	x_file = x_Trans.xmlToxml(txtXML.Text,txtXSL.Text);

	// To send the output to the container within the page
	txtResult.Text = x_file;
}
```

Since, the expected action of the Transform button was to get the result document; as a result, it is appropriate then to call upon the Web service in the code behind this button. After we instantiate the proxy class we pass the two strings we have collected from the txtXML, and the txtXSL text boxes, to the [WebMethod] xmlToxml() of the service. The x_file was declared globally as a string that will hold the result document. After we get the result we assign the x_file to txtResult text box. The property "Text" contains the unformatted text and is identical to the text entered in the control by the user. This property is not available at design time; read-only at run time.

The next fragment of code we will examine is the code behind the "Send" button.

```
private void Send_Click(object sender, System.EventArgs e)
{
	if (RadioButtonList1.SelectedItem.Value.Equals("File") &&
ResultFile.Text.EndsWith("xml"))
	{
		StreamWriter wr = new StreamWriter(ResultFile.Text);
		wr.Write(txtResult.Text);
		wr.Close();
	}
else if(RadioButtonList1.SelectedItem.Value.Equals("New_Browser_Page"))
	{
		// To send the output to the new Browser Window
		ASCIIEncoding ae = new ASCIIEncoding();
```

```
            byte[] byteXML = ae.GetBytes(txtResult.Text);

            Response.ContentType="text/xml";
            Response.OutputStream.Write(byteXML,0,byteXML.Length);
            Response.End();
      }
else if(RadioButtonList1.SelectedItem.Value.Equals("New_Windows_Container"))
      {
            // To send the output to a container out in a new page
            string ResultDoc = txtResult.Text;
            Response.Redirect("FormContainer.aspx?ResultDoc=" +
            System.Web.HttpUtility.UrlEncode(ResultDoc));
      }

}
```

After the user get the result document he/she will have three choices of what to do with the transformed XML document. These choices are in the form of radio buttons. Once the selection is made the user will click “Send”.

The code for the send button start with an if statement to examine if the radio button value was a “File” at the same time the name of the file need to be an .xml extension since we are expecting a transformed XML document. If that is the case then we ask the StreamWriter to write the content of the Result text box to the specified file name. StreamWriter class implements a TextWriter for writing characters to a stream in a particular encoding

The next branch of the if statement is the else if. This branch will examine if the selection value was "New_Browser_Page". In this case result will be opened in a new IE browser page. To establish this task we need to capture the string from the Result text box in to a byte array. But before we do that we have to associate an encoding with the string in this case is an ASCII encoding. The ASCIIEncoding class represents an ASCII character encoding of Unicode characters.

HttpResponse class encapsulates HTTP response information from an ASP.NET operation. One of the public properties of this class is the OutputStream, which enables binary output to the outgoing HTTP content body. But before we send the content in to an output stream we need to decide what the content type will be, in this case “text/xml”. The ContentType is one of the entity header fields of the HTTP header, which can be used in request messages or response messages. Also, they contain information about the entity-body of the message [12]. After we set the content type we will write the byteXML starting at potion 0 to the length of the byteXML.

The last else if statement will redirect the result to a new form in a different web page. In other word, the behavior will look like the new browser page but this time it will be a new text box in a different Web page. The Redirect() method is one of the public methods of the HttpResponse class that will redirects a client in our case the result document to a new URL. But before we redirect the result in to a new URL we have to construct the page that will contain the result.

The following code is the class ResultContainer() that sets behind the ASP.NET form for the new page.

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.IO;

namespace WebClientApplication
{
	public class ResultContainer : Page
	{
		public TextBox    ResultBox;
		public void Page_Load(Object Sender, EventArgs e)
		{
			if (!Page.IsPostBack)
			{
				ResultBox.Text = Request.Params["ResultDoc"];
			}
		}
	}
}
```

The Page class represents an .aspx file requested from a server that hosts an ASP.NET Web application. One of the public properties of the Page class is the IsPostBack which has the property value "true" if the page is being loaded in response to a client post back; otherwise, "false".

The ResultBox is our new text box that will contain the result for the transformed document. And the ResultDoc is the string passed from the txtResult of the main Web form of the client.

Now that we have examined codes for the main two buttons (Transform and Send); we would like to examine the code for the reset button, which is very straight forward. When the user clicks the "Reset" button the following code get executed:

```
private void Reset_Click(object sender, System.EventArgs e)
{
	txtXML.Text =" ";
	txtXSL.Text =" ";
	ResultFile.Text   = "";
	txtResult.Text = " ";
	RadioButtonList1.SelectedIndex = -1;
}
```

All we are doing is basically emptying each users entered text box or radio button selection. In the case of the radio button we are assigning -1 to the SelectedIndex property of the RadioButtonList1.

This will conclude the development of the Web client for the XML Transformer Web Service.

5.4 Building Console Client

The development of the Console based client for the XML Transformer Web Service is not different from the development of the Web client except we do not use any web components. It is purely a command line execution of the program. There will be only two augments one for the XML document and the other is for the XSLT document. The code for the console client is as follows:

```
using System;
// Import the XML Transformer Web Service
using ConsoleClientApplication.TransService;

namespace ConsoleClientApplication
{
	class ConsoleClient
	{
		[STAThread]
		static void Main(string[] args)
		{
			if (args.Length != 2)
			{
				Console.Write("Usage: ");
			Console.WriteLine("XmlDocument.xml XslDocument.xsl");
				return;
			}
			else
			{
				Trans x_Trans = new Trans();
			string x_file = x_Trans.xmlToxml(args[0],args[1]);

				Console.Write(x_file);
				Console.ReadLine();
			}
		}
	}
}
```

The if statement is there to check for the number of argument provided by the user. In case the user provides less than two arguments a “Usage:” message will be posted to inform the user with number and type of the files excepted.

The else part of the if statement is something we have seen in previous sections. When we get the string from the service basically we write it out to the console screen. The extra line of code

```
Console.ReadLine();
```

Is to allow us to read the result and hit “Enter” when we are done.

This will conclude the development of the Console client for the XML Transformer Web Service.

5.5 Testing and Running

In this section I will show some screen shots of the two clients. But before we will show the XML and XSLT document that we will be using.

The XML document:

```
<?xml version='1.0'?>
<!-- This file represents a fragment of a book store inventory database -->
<bookstore>
  <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
    <title>The Autobiography of Benjamin Franklin</title>
    <author>
      <first-name>Benjamin</first-name>
      <last-name>Franklin</last-name>
    </author>
    <price>8.99</price>
  </book>
  <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
    <title>The Confidence Man</title>
    <author>
      <first-name>Herman</first-name>
      <last-name>Melville</last-name>
    </author>
    <price>11.99</price>
  </book>
  <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
    <title>The Gorgias</title>
    <author>
      <name>Plato</name>
    </author>
    <price>9.99</price>
  </book>
</bookstore>
```

The XSLT document:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
    <root>
        <xsl:apply-templates/>
    </root>
    </xsl:template>
    <xsl:template match="bookstore">
    <!-- Prices and books -->
        <bookstore>
            <xsl:apply-templates select="book"/>
        </bookstore>
    </xsl:template>
    <xsl:template match="book">
        <book>
            <xsl:attribute name="ISBN">
                <xsl:value-of select="@ISBN"/>
            </xsl:attribute>
       <price><xsl:value-of select="price"/></price><xsl:text>
            </xsl:text>
        <title><xsl:value-of select="title"/></title><xsl:text>
```

```
            </xsl:text>
        </book>

    </xsl:template>
</xsl:stylesheet>
```

If you notice in the XSLT document we are only interested in the <book> element ISBN attribute and the elements <price> and <title>.

5.5.1 Testing and Running the Web Client

In figure 14 we have entered a URL for the XML document and the copy and pasted the XSLT document.

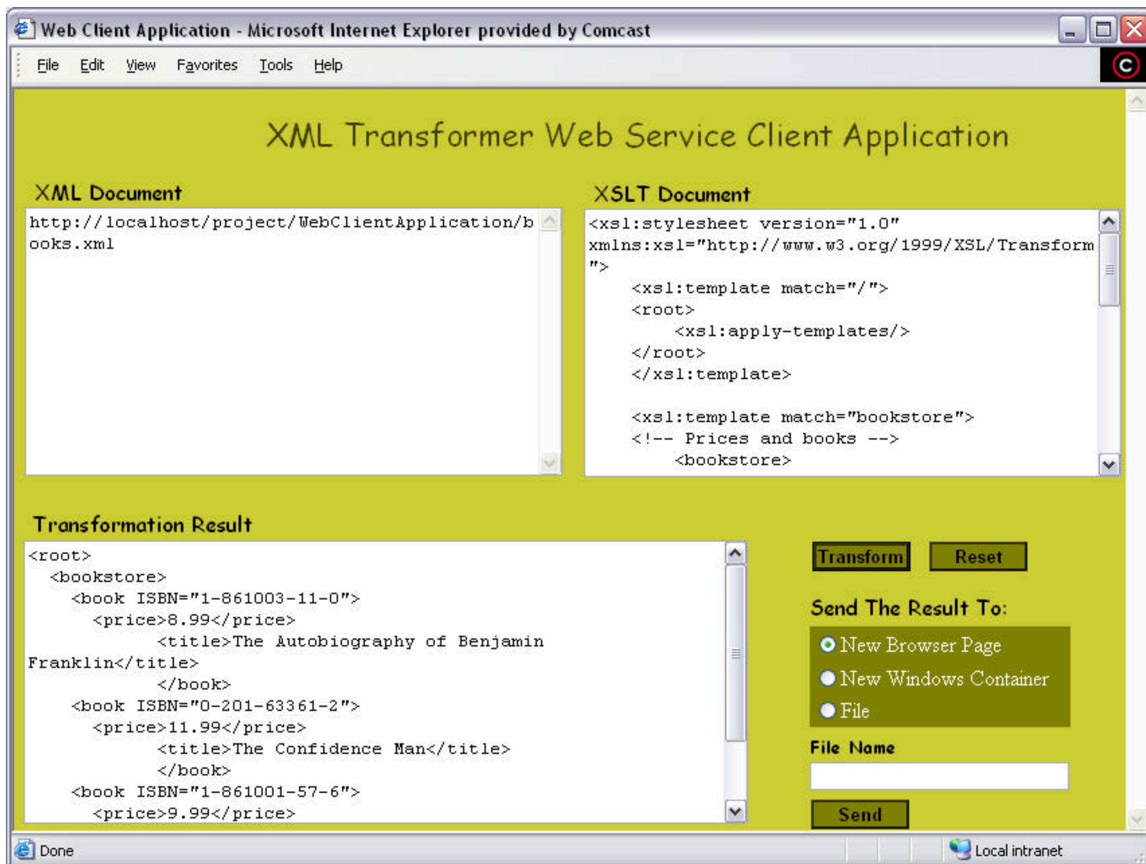

Figure 14

After we got the result we choose to send it to the a New Browser Page, Figure 15 shows the result in the IE browser

Figure 15

Figure 16 is the same result sent to a New Windows container

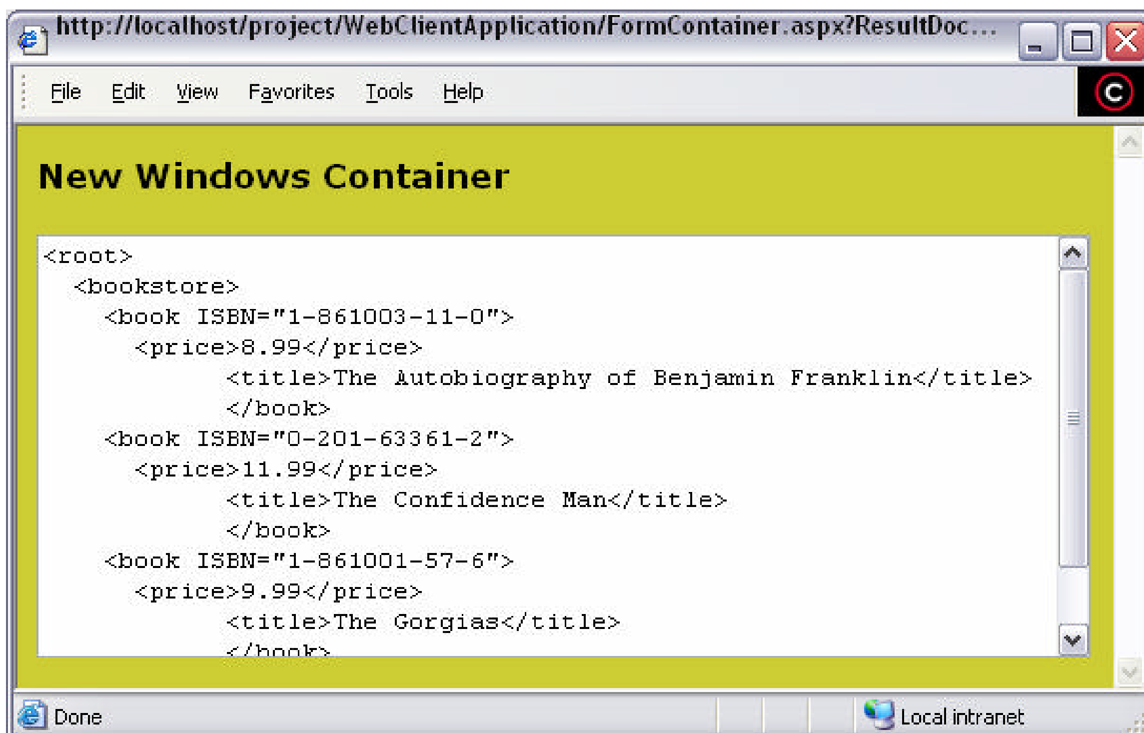

Figure 16

5.5.2 Testing and Running the Console Client

Figure 17 shows the console client that will run the following command line

>ConsoleClientApplication.exe C:\temp\books.xml
http://localhost/project/WebClientApplication/books.xsl

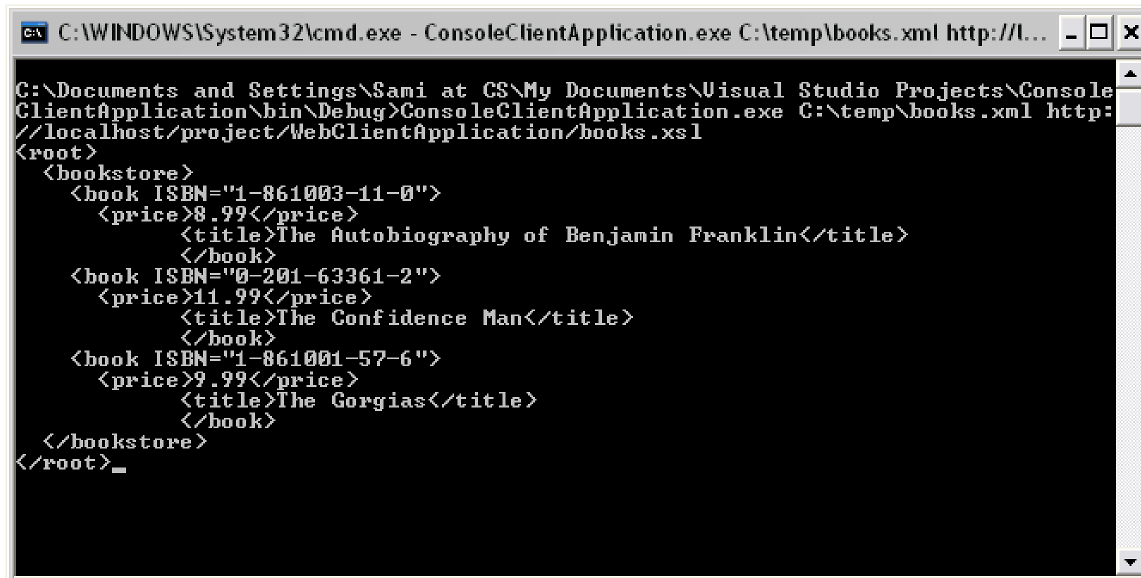

Figure 17

6.0 XML-SQL Application

In this section we will discusses the implementation of the XML-SQL Web application. This Web based application process XML documents that have SQL queries impeded inside them. To be able to run these queries across databases the connection string elements will also be provided. The application will collect these elements and send it over to the database. The result of the query will be in XML format that will replace the query elements and the string elements.

The main functionality of this application is to implement SAX (Simple API for XML) like parser using XmlReader. The XmlReader is a forward-only, read-only cursor. It provides fast, non-cached stream access to the input. It can read a stream or a document. It allows the user to pull data, and skip records of no interest to the application. The big difference lies in the fact that the SAX model is a "push" model, where the parser pushes events to the application, notifying the application every time a new node has been read, while applications using XmlReader can pull nodes from the reader at will [13]. Figure 18 illustrate the difference between the push model and the pull model.

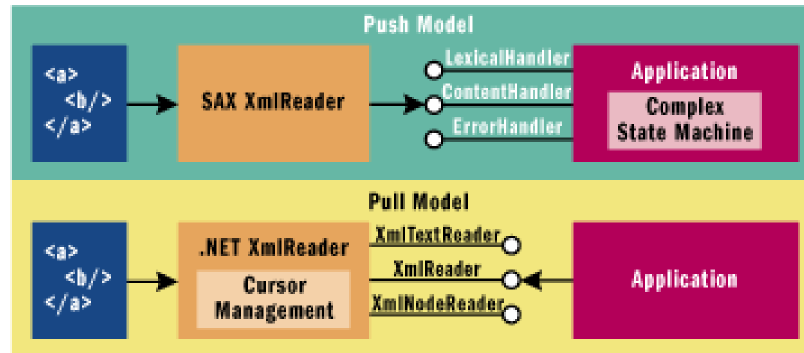

Figure 18 [14]

Also, this application will have the ability to communicate to more than one database based on the connection string provided. The input XML document can have other elements that are not in our interest but are relevant to the document as a whole. The tag <SQLStatement> will contain all the needed information for our application to connect to the database and return the result. The following XML fragment from the input document is an example of the <SQLStatement> tag:

```
<SQLStatement>
      <server>server = (local)\NetSDK</server>
      <security>Integrated Security = true</security>
      <database>database = Northwind</database>
      <timeout>Connection Timeout = 5</timeout>
      <DataSetElement>Employees_Table</DataSetElement>
      <ResultElement>Employees</ResultElement>
      <SQLQuery>SELECT EmployeeID, FirstName, LastName FROM
Employees</SQLQuery>
</SQLStatement>
```

The first four elements will be used as a connection string to the database. The method buildConnectionString() will build the connection string in this format:

```
server = (local)\NetSDK; Integrated Security = true; database = Northwind;
Connection Timeout = 5
```

It will add ";" after each string clause. The last string will not require a ";".

The <SQLQuery> will hold the query that will run across the database server and is required by the SqlDataAdapter class, which represents a set of data commands and a database connection that are used to fill the DataSet and update a SQL Server database. The <DataSetElement> and <ResultElement> is required by the DataSet class, which represents an in-memory cache of data. The <DataSetElement> will work as a root element for the whole result. And the <ResultElement> will work as a root element for each set. Finally, the <SQLStatement> will be replaced with <SQLQueryResult> also; the rest of the elements in between will be removed and replaced with result. The following XML fragment is the result of the above example:

```
<SQLQueryResult>
```

```
    <Employees_Table sql:query="SELECT EmployeeID, FirstName, LastName FROM
Employees" source="database = Northwind">
      <Employees>
        <EmployeeID>1</EmployeeID>
        <FirstName>Nancy</FirstName>
        <LastName>Davolio</LastName>
      </Employees>
       ...................................
      <Employees>
        <EmployeeID>2</EmployeeID>
        <FirstName>Andrew</FirstName>
        <LastName>Fuller</LastName>
      </Employees>
    </Employees_Table>
  </SQLQueryResult>
```

We notice that the `<DataSetElement>Employees_Table</DataSetElement>` and `<ResultElement>Employees</ResultElement>` are part of the result in the same way we described above. The `<DataSetElement>` included two attributes the `sql:query` and `source` these two attributes are not required by the result document or the DataSet class, I have added them for more clarity so we would know where we are getting the result from and what was the query that generated the result.

In the next section we will discuss the code that will read these elements and extract the text behind them.

6.1 Building Web Based Application

In this section we will discuss the code behind XML-SQL Web application. However, we will not cover all of the code since some of the code where discussed in previous sections. Also, we will not discuses the ASP.NET component in this application since they are the same component used in the previous application. We will start with the ASP.NET form that worked as an interface to this application. Figure 19 is the Web form for the XML-SQL application.

We will start with the Run_Click() method which is the method that sets behind the "Run" button and is the main method in this application. The main data structure in this method is the Stack, while loop and switch statement. After we prepare the XmlTextWriter to write to stream, as we did in the XML Transformer Web Service, we will ask the XmlTextReader to read the input document. Since we expect the document to be coming from file, URL, or memory stream we used the same method but different name we implemented earlier to read the input document. This method is the ReadXmlSqlTextBox()

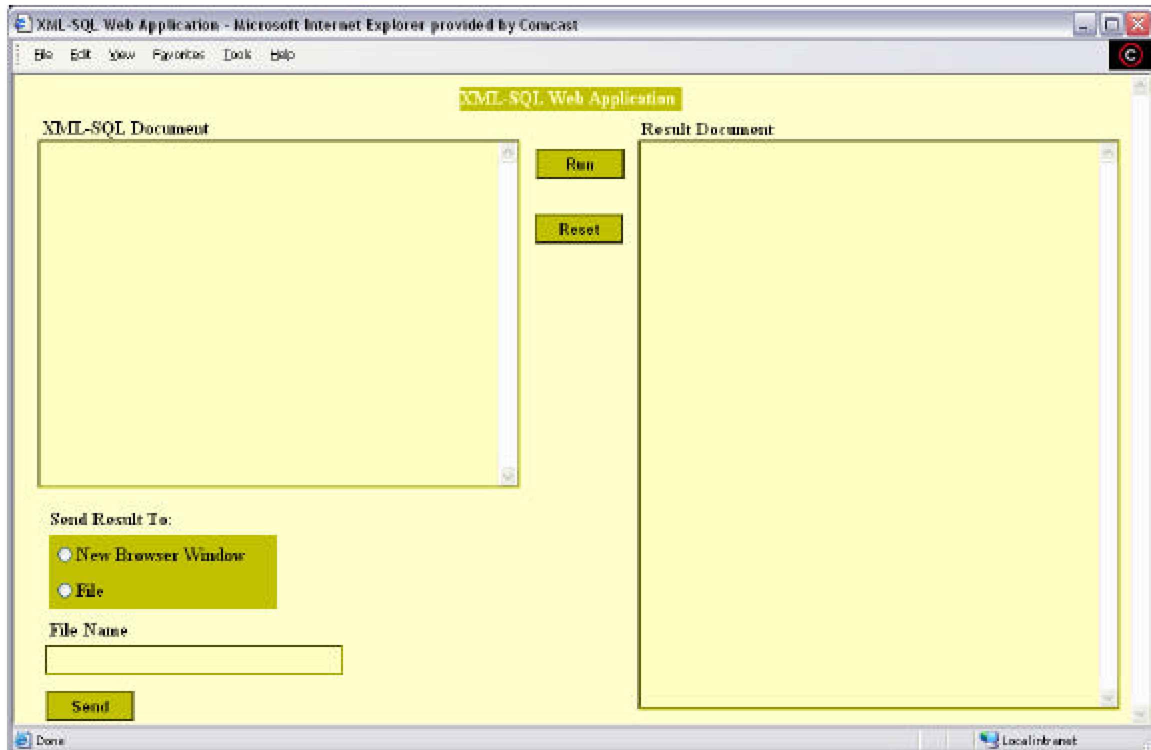

Figure 19

```
private void Run_Click(object sender, System.EventArgs e) {

            //XML Writer send the data to a Memory Stream
            XmlTextWriter myWriter;
            Stream s = new MemoryStream();
            myWriter = new XmlTextWriter(s, null);
            myWriter.Formatting = Formatting.Indented;

            //Calling the method that will read the XML-SQL Document Text Box
            //to find out wither it is a Document or a file
            XmlTextReader myReader = ReadXmlSqlTextBox();
            myReader.WhitespaceHandling = WhitespaceHandling.Significant;
```

Since we are implementing SAX like model we want to react to elements as they appear or come to us. We will not pull the elements from the documents. The way we will implement this is by a while loop that will loop as long as the reader object is reading. In this case myReader.Read() then we implement a switch statement that will read each XML node type once we read an element we push it to the stack. We will start with first case which is the

```
Stack myStack = new Stack();
// Reading and Writing Loop
while(myReader.Read()) {
      switch(myReader.NodeType) {
            case XmlNodeType.Element:
                  if(myReader.Name == "server"){
                        myStack.Push(myReader.Name);
                  }
```

```
                                        else if(myReader.Name == "security"){
                                              myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name == "database"){
                                              myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name == "timeout"){
                                              myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name=="DataSetElement")
{
                                              myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name =="ResultElement")
{
                                              myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name=="SQLQuery") {
                                              myStack.Push(myReader.Name);
                                        }
                                  else if (myReader.Name == "SQLStatement"){
                                  myStack.Push(myReader.Name);
                                  myWriter.WriteStartElement("SQLQueryResult");
                                  myWriter.WriteAttributes(myReader, true);
                                        }
                                        else {
                                  myStack.Push(myReader.Name);
                                  myWriter.WriteStartElement(myReader.Name);
                                  myWriter.WriteAttributes(myReader, true);
                                        }
                                  break;
```

As we are reading through the input document we may come across elements that are not in our interest. We will need to write those elements as they appear to the reader to the output document. The last else statement will do this function. However, if we come across elements that are important to the application we do not want to write them but consume them and push them to the stack. The if else statement before the end will replaced `<SQLStatement>` with `<SQLQueryResult>`.

The other important case in this switch statement is the `case XmlNodeType.EndElement:` since we do not want to write those elements that are used to build the connection string and the elements needed by the DataSet class and the SqlDataAdapter class, once we get to the end element we will pop our stack for any elements. The if statement in this switch case will filter those elements and the else statement will write everything else.

```
                                  case XmlNodeType.CDATA:
                                        myWriter.WriteCData(myReader.Value);
                                        break;

                                  case XmlNodeType.EndElement:
if ((myReader.Name== "DataSetElement") || (myReader.Name== "ResultElement")
|| (myReader.Name== "SQLQuery")||(myReader.Name== "server")||
(myReader.Name== "security") || (myReader.Name==
"database")||(myReader.Name== "timeout")) {
                                              myStack.Pop();
```

```
                                    }
                                    else {
                                          myWriter.WriteEndElement();
                                          myStack.Pop();
                                    }
                                    break;

                              case XmlNodeType.EntityReference:
                                    myWriter.WriteEntityRef(myReader.Name);
                                    break;

                              case XmlNodeType.ProcessingInstruction:
      myWriter.WriteProcessingInstruction(myReader.Name,myReader.Value);
                                    break;
```

The other important case is the `case XmlNodeType.Text:` in this case we will be extracting the values that are important to the application. This case was implement by another method

```
                        case XmlNodeType.Text:
                        // Call the method that will peek in top of the stack
                        // and extract the DB elements
                        ExtractDBElements(myStack, myReader, myWriter);
                        break;
```

The ExtractDBElements() will talk three parameters the stack object the reader and the writer object as input parameters. And it will return void. The method will start with if statement for each element by peeking at the top of the stack. If at the top of the stack is the value that we are interested in then will take that value and assign it to a global variable. For example, `DBServer = myReader.Value` DBserver is a global variable for the database server that we would like to connect to. The code for the ExtractDBElements()is as follows:

```
private void ExtractDBElements(Stack myStack, XmlTextReader myReader,
XmlTextWriter myWriter){

                  if (myStack.Peek().Equals("server")) {
                        DBServer = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("security")) {
                        DBSecurity = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("database")) {
                        Database = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("timeout")) {
                        DBTimeout = myReader.Value;
                        // build the Connection String
                        buildConnectionString();
                  }
                  else if (myStack.Peek().Equals("DataSetElement")) {
                        DataSetValue = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("ResultElement")) {
                        ResultSetValue = myReader.Value;
```

```
                  }
                  else if (myStack.Peek().Equals("SQLQuery")) {
                        SQLQueryElement = myReader.Value;

                  // Connect to Database and return the Result in XML Format
                        string ResultXML = DatabaseConnection();

      // Crate an XmlTextReader object Reader1 to Read the string from DB
                        ASCIIEncoding AEnCod = new ASCIIEncoding();
                        byte[] XMLbyte = AEnCod.GetBytes(ResultXML);
                        MemoryStream myStr = new MemoryStream(XMLbyte);
                        XmlTextReader Reader1 = new XmlTextReader(myStr);

                        while(Reader1.Read()) {
// Reading and Write the Result in between the SQLStatament Element/Tag
                  ReadingWritingDataSet(Reader1, myWriter, DataSetValue);
                        }
                        Reader1.Close();
                        myStr.Close();
                  }// Close the if for the "SQLQuery" Element
                  else {
                        myWriter.WriteString(myReader.Value);
                  }
            }
```

Notice after we peek for the time out string we call buildConnectionString() method that is because at this point we have collected all the information that we want use for connecting to the server. We will continue reading the rest of the elements until we reach `SQLQuery` element again at this point we know that we have collected every thing we need for us to be able to connect to the database and the requirements for the DataSet class and the SqlDataAdapter class. We will call the DatabaseConnection() method, which will return for us the result in XML format. After we get the result we assign it to local string and start reading the and writing this result using a new reader object but the same writer object. That is because we want to replace the seven elements with the result that come from the database.

For this purpose we created a new method that will take care of the reading the result and writing it to the out put document. The last else statement in the ExtractDBElements() method will write to the output document every thing else that was not relevant to connect to the database or to run the query.

We will talk now about the ReadingWritingDataSet() method. This method as we said will write the result that we got from the database. We start our while loop before we call the method and the method itself will implement the switch case. In this case it will be the same switch case that we used above the only difference will be is that we are writing extra attributes to the dataset element. The reason for these attributes is when we write the final document we would like to know the query that generated this result and the source of this result. The following code will accomplish this task:

```
if (Reader1.Name == DataSet) {
      // Writing the Query statment as an attrbute element
      // <sql:query> in the DataSet Element
      myWriter.WriteStartElement(Reader1.Name);
```

```
myWriter.WriteAttributeString("sql:query",SQLQueryElement);
// Writing the Database Name as a attrbuite element
// <from> in the DataSet Element
myWriter.WriteAttributeString("source",Database);
}
```

Now we would like to discusses the DatabaseConnection() method. In this method we establish connection with the database and pass the query and the connection string to the SqlDataAdapter class and pass the dataset element and the result element to the DataSet class.

```
private string DatabaseConnection() {

    // Conecting to the Database
    string conStr = ConnectionString;

    // Connecting to Database and Open the Connection
    SqlConnection sqlConn = new SqlConnection(conStr);
    sqlConn.Open();

    // Running the Query
    SqlDataAdapter da = new SqlDataAdapter(SQLQueryElement, sqlConn);

    // Passing the DataSet Element
    DataSet ds = new DataSet(DataSetValue);

    // Passing the Result Element
    da.Fill(ds, ResultSetValue);
```

The method WriteXml() will Writes XML data, and optionally the schema, from the DataSet. We would like to capture the result in to string so we instantiate a string builder and use TextWriter class that will hold the result in the object sw. When we are done we read the object using the BuildResult object that was crated by StringBuilder class and return the string to the ExtractDBElements() method.

```
    // Instantiate a string builder to hold the XML Result in to a string
    StringBuilder BuildResult = new StringBuilder();
    TextWriter sw = (TextWriter) new StringWriter(BuildResult, null);

    // Writing the Result in XML Format
    ds.WriteXml(sw);
    sqlConn.Close();

    string ResultDoc = BuildResult.ToString();

    return ResultDoc;
}
```

Now we go back to out Run_Click() method. We can read our writer object and write the result in to the result text box. To better see the result we have used the same Send_Click() method we used previously.

In the next section we will run this application and test its functionality.

6.3 Testing and Running

Before we start out test case I would like to introduce the database software I used with this application. I used Microsoft Data Engine (MSDE) 7.0 which is a scaled version of Microsoft SQL Server that sets in top of Microsoft Access. This database engine was installed by VS.NET it was one of the options you have to choose it will not be installed automatically. It is also can be downloaded for free from Microsoft download page. It will come with pre configured databases as examples to work on.

The input document I used is:

```
<?xml version="1.0"?>
<ROOT xmlns:3ql="urn:schemas-microsoft-com:xml-sql">
<E1> Any other Elements, and as many as we want </E1>
<SQLStatement>
<server>server = (local)\NetSDK</server>
<security>Integrated Security = true</security>
<database>database = Northwind</database>
<timeout>Connection Timeout = 5</timeout>
<DataSetElement>Employees_Table</DataSetElement>
<ResultElement>Employees</ResultElement>
<SQLQuery>SELECT EmployeeID, FirstName, LastName FROM Employees</SQLQuery>
</SQLStatement>
<E2> Any other Elements, and as many as we want </E2>
<SQLStatement>
<server>server = (local)\NetSDK</server>
<security>Integrated Security = true</security>
<database>database = GrocerToGo</database>
<timeout>Connection Timeout = 5</timeout>
<DataSetElement>Categories_Table</DataSetElement>
<ResultElement>Categories</ResultElement>
<SQLQuery>SELECT CategoryID, CategoryName FROM Categories</SQLQuery>
</SQLStatement>
<E3> Any other Elements, and as many as we want </E3>
<SQLStatement>
<server>server = (local)\NetSDK</server>
<security>Integrated Security = true</security>
<database>database = pubs</database>
<timeout>Connection Timeout = 5</timeout>
<DataSetElement>authors_Table</DataSetElement>
<ResultElement>Authors</ResultElement>
<SQLQuery>SELECT au_id, au_lname, au_fname FROM authors</SQLQuery>
</SQLStatement>
<E4> Any other Elements, and as many as we want </E4>
</ROOT>
```

We can see that we are connecting to three different databases and running three different queries. All of these databases are available in the MSDE.

After we run the application the result document will look like this:

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <E1> Any other Elements, and as many as we want </E1>
  <SQLQueryResult>
    <Employees_Table sql:query="SELECT EmployeeID, FirstName, LastName FROM
Employees" source="database = Northwind">
      <Employees>
        <EmployeeID>1</EmployeeID>
        <FirstName>Nancy</FirstName>
        <LastName>Davolio</LastName>
      </Employees>
      <Employees>
        <EmployeeID>2</EmployeeID>
        <FirstName>Andrew</FirstName>
        <LastName>Fuller</LastName>
      </Employees>
      ................................
      <Employees>
        <EmployeeID>9</EmployeeID>
        <FirstName>Anne</FirstName>
        <LastName>Dodsworth</LastName>
      </Employees>
    </Employees_Table>
  </SQLQueryResult>
  <E2> Any other Elements, and as many as we want </E2>
  <SQLQueryResult>
    <Categories_Table sql:query="SELECT CategoryID, CategoryName FROM
Categories" source="database = GrocerToGo">
      <Categories>
        <CategoryID>3</CategoryID>
        <CategoryName>Soda</CategoryName>
      </Categories>
      <Categories>
        <CategoryID>2</CategoryID>
        <CategoryName>Cereal</CategoryName>
      </Categories>
      <Categories>
        <CategoryID>1</CategoryID>
        <CategoryName>Milk</CategoryName>
      </Categories>
    </Categories_Table>
  </SQLQueryResult>
  <E3> Any other Elements, and as many as we want </E3>
  <SQLQueryResult>
    <authors_Table sql:query="SELECT au_id, au_lname, au_fname FROM authors"
source="database = pubs">
      <Authors>
        <au_id>409-56-7008</au_id>
        <au_lname>Bennet</au_lname>
        <au_fname>Abraham</au_fname>
      </Authors>
      ................................
      <Authors>
        <au_id>672-71-3249</au_id>
        <au_lname>Yokomoto</au_lname>
```

```
        <au_fname>Akiko</au_fname>
      </Authors>
    </authors_Table>
  </SQLQueryResult>
  <E4> Any other Elements, and as many as we want </E4>
</ROOT>
```

Since we are interested in the functionality rather than the result I have collapsed some of the result and placed .... to show that it is a continues.

7.0 Conclusion

In this report we have talked about the XML Web Service in general and the building blocks of any XML Web Service: SOAP, WSDL, and UDDI. Also, we have seen how to build an XML Web Service in .NET Environment. The [WebService] attributes, which is placed in top of the class, is an optional attributes that can have the service name and description. The [WebMethod] attribute is required to be placed in top of any method that we would like to access over the web or want to invoke its capability using a consumer.

The XML Transformer Web Service was capable of transforming the XML document in to another XML document according to a given XSLT document. The input streams to the service could be in any format. For example, the XML file could be coming from the Internet using a URL and the XSLT file could be coming from a memory stream. The service will be able to handle mixed inputs streams. We have also, built both Web based and Console based client that will work as an interface to our service. The Web based client involved more programming than the console based since we are using ASP.NET Web Components. When we built the Web based client we considered the user friendliness factor by giving the user better control over the transformed XML document.

The XML-SQL Web based application was capable of taking XML Template document which contains SQL queries and connection string to the database. Using the connection string the application will run the query and return the result in XML format. We have used some of the ASP.NET Web Component such as radio buttons.

8.0 References

[1] Dalvi, D., Gray, J., Joshi, B., Normen, F., Norton, F., Olsen, A., Palermo IV, M., Singh, D., Slater, J., & Williams, K., (December 2001). Professional XML for .NET Developers. Chicago, Illinois: Wrox Press, Inc.

[2] Short, Scott, (2002) Building XML Web Services for the Microsoft .NET Platform. Redmond, Washington: Microsoft Press.

[3] Freeman, A. & Jones, A., (2003) Microsoft .NET XML Web Services Step by Step. Redmond, Washington: Microsoft Press.

[4] Microsoft .NET web site. *What are XML Web Services?*, URL: http://www.microsoft.com/net/basics/xmlservices.asp, (January 14, 2002)

[5] Wolter, Roger *XML Web Services Basics,* URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/webservbasics.asp (December 2001)

[6] Tapang, Carlos C. *Web Services Description Language (WSDL) Explained*, URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsdlexplained.asp (July 2001)

[7] Wyke, A., Rehman, S., & Leupen, B., (2002). XML Programming. Redmond, Washington: Microsoft Press.

[8] Deitel, H. M., Deitel, P. J., & Nieto, T. R. (2001) Internet and World Wide Web How to program. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

[9] Visual Studio .NET Help pages for "XSLT Transformations with the XslTransform Class"

[10] Visual Studio .NET Help pages for "XPthDocument"

[11] Visual Studio .NET Help pages for "XmlTextWriter"

[12] Visual Studio .NET Help pages for "HTTP Headers Reference"

[13] Visual Studio .NET Help pages for "Comparing XmlReader to SAX Reader"

[14] Skonnard, Aaron. *XML in .NET: .NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation*, (January 2001) MSDN Magazine

Appendix A: Files Used and Descriptions

Table 1: Lists the Web Reference directory auto generated files that correspond to any client communicating with XML Web Service:

File Name	Description
Reference.map	An XML file that maps the URL of the XML Web Service WSDL and Discovery (DISCO) files to the locally cached versions of the files (listed below). Maintain a reference to the original file files allows Visual Studio .NET to easily regenerate the proxy if the service description changes.
Reference.cs	The C# source file that contains the auto generated code for the proxy class.
<Filename>.disco	A locally cashed copy of the discovery file for the .NET XML Web Service.
<Filename>.wsdl	A locally cashed copy of the discovery file for the XML Web Service's WSDL service description file. The name of the file depends on the XML Web Service. By default, the file will have the same name of the service's .asmx file but with a .wsdl extension

Table 2: Lists the files in the main directory of the .NET XML Web Service and ASP.NET Web applications:

File Name	Description
Assembly.cs	Contains metadata that will applied to the assemblies contained in a project, including name, version, and culture information.
Global.asax	This is also known as "ASP.NET application file" allows the developer to add application and session startup procedures and event handlers. The functionality are implemented in a separate file with an addition of the language extension such as .cs for "C#" of .vb for "Visual Basic"
Global.asax.cs	C# code that contain the functionality of the Global.asax
Global.asax.resx	A file used to edit and define application resources for the Global.asax file. It is in XML Format

<Filename>.asmx	Contain the main logic of .NET XML Web Service. It has the ASP.NET directives indicating that this file represent an XML Web Service. The functionality are implemented in a separate file with an addition of the language extension such as .cs for "C#" of .vb for "Visual Basic"
<Filename>.asmx.cs	C# code that contain the functionality of the .NET XML Web Service files .asmx
<Filename>.asmx.resx	A file used to edit and define application resources for the .asmx file. It is in XML Format
<Filename>.vsdisco	This file is used by ASP.NET to auto generate DISCO files from XML Web Service.
<Filename>.aspx	ASP.NET Web form. The functionality are implemented in a separate file with an addition of the language extension such as .cs for "C#" of .vb for "Visual Basic"
<Filename>.aspx.cs	C# code that contain the functionality of the ASP.NET Web form files .aspx
Web.config	Is an XML file that contains configuration settings for XML Web Service

Appendix B: XML Classes in .NET

XML Classes that are available under in .NET are under System.Xml namespace, which provides standards-based support for processing XML.

Class	Description
NameTable	Implements a single-threaded XmlNameTable.
XmlAttribute	Represents an attribute. Valid and default values for the attribute are defined in a DTD or schema.
XmlAttributeCollection	Represents a collection of attributes that can be accessed by name or index.
XmlCDataSection	Represents a CDATA section.
XmlCharacterData	Provides text manipulation methods that are used by several classes.
XmlComment	Represents the content of an XML comment.
XmlConvert	Encodes and decodes XML names and provides methods for converting between common language runtime types and XML Schema definition language (XSD) types. When converting data types the values returned are locale independent.
XmlDataDocument	Allows structured data to be stored, retrieved, and manipulated through a relational DataSet.
XmlDeclaration	Represents the XML declaration node: <?xml version='1.0' ...?>.
XmlDocument	Represents an XML document.
XmlDocumentFragment	Represents a lightweight object that is useful for tree insert operations.
XmlDocumentType	Represents the document type declaration.
XmlElement	Represents an element.
XmlEntity	Represents an entity declaration: <!ENTITY ... >.
XmlEntityReference	Represents an entity reference node.
XmlException	Returns detailed information about the last exception.
XmlImplementation	Defines the context for a set of XmlDocument objects.
XmlLinkedNode	Gets the node immediately preceding or following this node.
XmlNamedNodeMap	Represents a collection of nodes that can be accessed by name or index.
XmlNamespaceManager	Resolves, adds and removes namespaces to a collection and provide scope management for these namespaces. This class is used by the XsltContext and XmlReader classes.
XmlNameTable	Table of atomized string objects.
XmlNode	Represents a single node in the XML document.
XmlNodeChangedEventArgs	Provides data for the NodeChanged, NodeChanging, NodeInserted, NodeInserting, NodeRemoved and NodeRemoving events.

XmlNodeList	Represents an ordered collection of nodes.
XmlNodeReader	Represents a reader that provides fast, non-cached forward only access to XML data in an XmlNode .
XmlNotation	Represents a notation declaration: <!NOTATION ... >.
XmlParserContext	Provides all the context information required by XmlTextReader or XmlValidatingReader to parse an XML fragment.
XmlProcessingInstruction	Represents a processing instruction, which XML defines to keep processor-specific information in the text of the document.
XmlQualifiedName	Represents an XML qualified name.
XmlReader	Represents a reader that provides fast, non-cached, forward-only access to XML data.
XmlResolver	Resolves external XML resources named by a URI.
XmlSignificantWhitespace	Represents white space between markup in a mixed content mode or white space within an xml:space= 'preserve' scope. This is also referred to as significant white space.
XmlText	Represents the text content of an element or attribute.
XmlTextReader	Represents a reader that provides fast, non-cached, forward-only access to XML data.
XmlTextWriter	Represents a writer that provides a fast, non-cached, forward-only way of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.
XmlUrlResolver	Resolves external XML resources named by a URI.
XmlValidatingReader	Represents a reader that provides DTD, XML-Data Reduced (XDR) schema, and XML Schema definition language (XSD) schema validation.
XmlWhitespace	Represents white space in element content.
XmlWriter	

Appendix C: SQL Classes in .NET

SQL Classes that are available in .NET are under the System.Data.SqlClient namespace, which is the SQL Server .NET Data Provider.

A .NET data provider describes a collection of classes used to access a SQL Server database in the managed space.

Class	Description
SqlClientPermission	Provides the capability for the SQL Server .NET Data Provider to ensure that a user has a security level adequate to access a data source.
SqlClientPermissionAttribute	Associates a security action with a custom security attribute.
SqlCommand	Represents a Transact-SQL statement or stored procedure to execute against a SQL Server database. This class cannot be inherited.
SqlCommandBuilder	Provides a means of automatically generating single-table commands used to reconcile changes made to a DataSet with the associated SQL Server database. This class cannot be inherited.
SqlConnection	Represents an open connection to a SQL Server database. This class cannot be inherited.
SqlDataAdapter	Represents a set of data commands and a database connection that are used to fill the DataSet and update a SQL Server database. This class cannot be inherited.
SqlDataReader	Provides a means of reading a forward-only stream of rows from a SQL Server database. This class cannot be inherited.
SqlError	Collects information relevant to a warning or error returned by SQL Server. This class cannot be inherited.
SqlErrorCollection	Collects all errors generated by the SQL .NET Data Provider. This class cannot be inherited.
SqlException	The exception that is thrown when SQL Server returns a warning or error. This class cannot be inherited.
SqlInfoMessageEventArgs	Provides data for the InfoMessage event. This class cannot be inherited.
SqlParameter	Represents a parameter to a SqlCommand, and optionally, its mapping to DataSet columns. This class cannot be inherited.
SqlParameterCollection	Collects all parameters relevant to a SqlCommand as well as their respective mappings to DataSet columns. This class cannot be inherited.
SqlRowUpdatedEventArgs	Provides data for the RowUpdated event. This class cannot be inherited.
SqlRowUpdatingEventArgs	Provides data for the RowUpdating event. This class cannot be inherited.
SqlTransaction	Represents a Transact-SQL transaction to be made in a SQL Server database. This class cannot be inherited.

Appendix D: Source Code Attachments

File Name: Trans.asmx.cs
For: XML Transformer Web Service

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Xml;
using System.Xml.Xsl;
using System.Xml.XPath;
using System.Text;
using System.IO;

namespace Transformer
{
	[WebService(Namespace="http://localhost/project/XMLTrans/",
	      Name="Trans",
	      Description="XML Web Service for Transforming XML to XML
Document.")]
	public class Trans : System.Web.Services.WebService
	{
		public Trans()
		{
			//CODEGEN: This call is required by the ASP.NET Web
Services Designer
			InitializeComponent();
		}
		#region Component Designer generated code

		//Required by the Web Services Designer
		private IContainer components = null;

		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
		}

		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if(disposing && components != null)
			{
				components.Dispose();
```

```
			}
			base.Dispose(disposing);
		}

		#endregion

		[WebMethod]
		public string xmlToxml(string xmlinput, string xslinput)
		{
			//Load the XML data file
			XPathDocument xmlDoc = new
XPathDocument(ReadInputDocument(xmlinput));

			//Create the XslTransform and load the stylesheet
			XslTransform xslDoc = new XslTransform();
			xslDoc.Load(ReadInputDocument(xslinput));

			//Create an XmlTextWriter to handle the output.
			XmlTextWriter myWriter;

			//XML Writer send the data to a Memory Stream
			Stream s = new MemoryStream();
			myWriter = new XmlTextWriter(s, null);
			myWriter.Formatting = Formatting.Indented;

			//Transform the file.
			xslDoc.Transform(xmlDoc, null, myWriter);

			// Reading the Tranfomed document from the Memeory Stream
			// Then returing the string to the client/consumer
			myWriter.Flush();
			s.Position = 0;
			StreamReader sr = new StreamReader(s);
			return sr.ReadToEnd();

			myWriter.Close();

		}

		private XmlTextReader ReadInputDocument(string Doc)
		{
			XmlTextReader DocReader;

			if (Doc.EndsWith("xml") || Doc.EndsWith("xsl"))
			{
				DocReader = new XmlTextReader(Doc);
			}
			else
			{
				// Memory stream reading the XML Document/Data
				byte[] byteDoc = Encoding.UTF8.GetBytes(Doc);
				MemoryStream DocStream = new MemoryStream(byteDoc);
				DocReader = new XmlTextReader(DocStream);
			}
			return DocReader;

		   DocReader.Close();
```

```
            }
        }
}
```

File Name: Trans.wsdl
For: XML Transformer Web Service

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://localhost/project/XMLTrans/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://localhost/project/XMLTrans/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://localhost/project/XMLTrans/">
      <s:element name="xmlToxml">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="xmlinput"
type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="xslinput"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="xmlToxmlResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="xmlToxmlResult"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </types>
  <message name="xmlToxmlSoapIn">
    <part name="parameters" element="s0:xmlToxml" />
  </message>
  <message name="xmlToxmlSoapOut">
    <part name="parameters" element="s0:xmlToxmlResponse" />
  </message>
  <message name="xmlToxmlHttpGetIn">
    <part name="xmlinput" type="s:string" />
    <part name="xslinput" type="s:string" />
  </message>
  <message name="xmlToxmlHttpGetOut">
    <part name="Body" element="s0:string" />
  </message>
  <message name="xmlToxmlHttpPostIn">
    <part name="xmlinput" type="s:string" />
```

```
    <part name="xslinput" type="s:string" />
  </message>
  <message name="xmlToxmlHttpPostOut">
    <part name="Body" element="s0:string" />
  </message>
  <portType name="TransSoap">
    <operation name="xmlToxml">
      <input message="s0:xmlToxmlSoapIn" />
      <output message="s0:xmlToxmlSoapOut" />
    </operation>
  </portType>
  <portType name="TransHttpGet">
    <operation name="xmlToxml">
      <input message="s0:xmlToxmlHttpGetIn" />
      <output message="s0:xmlToxmlHttpGetOut" />
    </operation>
  </portType>
  <portType name="TransHttpPost">
    <operation name="xmlToxml">
      <input message="s0:xmlToxmlHttpPostIn" />
      <output message="s0:xmlToxmlHttpPostOut" />
    </operation>
  </portType>
  <binding name="TransSoap" type="s0:TransSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <operation name="xmlToxml">
      <soap:operation soapAction="http://localhost/project/XMLTrans/xmlToxml"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <binding name="TransHttpGet" type="s0:TransHttpGet">
    <http:binding verb="GET" />
    <operation name="xmlToxml">
      <http:operation location="/xmlToxml" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <binding name="TransHttpPost" type="s0:TransHttpPost">
    <http:binding verb="POST" />
    <operation name="xmlToxml">
      <http:operation location="/xmlToxml" />
      <input>
        <mime:content type="application/x-www-form-urlencoded" />
      </input>
      <output>
```

```
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <service name="Trans">
    <documentation>XML Web Service for Transforming XML to XML 
Document.</documentation>
    <port name="TransSoap" binding="s0:TransSoap">
     <soap:address location="http://localhost/project/XMLTrans/Trans.asmx" />
    </port>
    <port name="TransHttpGet" binding="s0:TransHttpGet">
     <http:address location="http://localhost/project/XMLTrans/Trans.asmx" />
    </port>
    <port name="TransHttpPost" binding="s0:TransHttpPost">
     <http:address location="http://localhost/project/XMLTrans/Trans.asmx" />
    </port>
  </service>
</definitions>
```

File Name: WebForm.aspx.cs
For: The Web Based Client for the XML Transformer Web Service

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.IO;
using System.Xml;
using System.Xml.Xsl;
using System.Xml.XPath;
using System.Text;
// Importing the XML Transfomer Web Service
using WebClientApplication.TransService;

namespace WebClientApplication
{
	/// <summary>
	/// Summary description for WebForm.
	/// </summary>
	public class WebForm : System.Web.UI.Page
	{
		protected System.Web.UI.WebControls.TextBox txtXML;
		protected System.Web.UI.WebControls.Label Label1;
		protected System.Web.UI.WebControls.Label Label2;
		protected System.Web.UI.WebControls.TextBox txtXSL;
		protected System.Web.UI.WebControls.TextBox txtResult;
		protected System.Web.UI.WebControls.Label Label3;
		protected System.Web.UI.WebControls.Label Label4;
		protected System.Web.UI.WebControls.Label Label5;
```

```
		protected System.Web.UI.WebControls.TextBox ResultFile;
		protected System.Web.UI.WebControls.Label Label6;
		protected System.Web.UI.WebControls.Button Transform;
		protected System.Web.UI.WebControls.RadioButtonList 
RadioButtonList1;
		protected System.Web.UI.WebControls.Button Send;
		protected System.Web.UI.WebControls.Button Reset;

		private string x_file;

		private void Page_Load(object sender, System.EventArgs e)
		{
			// Put user code to initialize the page here
		}

		#region Web Form Designer generated code
		override protected void OnInit(EventArgs e)
		{
			//
			// CODEGEN: This call is required by the ASP.NET Web Form 
Designer.
			//
			InitializeComponent();
			base.OnInit(e);
		}

		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.txtXML.TextChanged += new 
System.EventHandler(this.txtXML_TextChanged);
			this.txtXSL.TextChanged += new 
System.EventHandler(this.txtXSL_TextChanged);
			this.txtResult.TextChanged += new 
System.EventHandler(this.txtResult_TextChanged);
			this.ResultFile.TextChanged += new 
System.EventHandler(this.ResultFile_TextChanged);
			this.Transform.Click += new 
System.EventHandler(this.Transform_Click);
			this.Reset.Click += new 
System.EventHandler(this.Reset_Click);
			this.RadioButtonList1.SelectedIndexChanged += new 
System.EventHandler(this.RadioButtonList1_SelectedIndexChanged);
			this.Send.Click += new 
System.EventHandler(this.Send_Click);
			this.Load += new System.EventHandler(this.Page_Load);

		}
		#endregion

		private void txtXML_TextChanged(object sender, System.EventArgs 
e)
		{
```

```
            }

            private void txtXSL_TextChanged(object sender, System.EventArgs
e)
            {

            }

            private void txtResult_TextChanged(object sender,
System.EventArgs e)
            {

            }

            private void RadioButtonList_SelectedIndexChanged(object sender,
System.EventArgs e)
            {

            }

            private void ResultFile_TextChanged(object sender,
System.EventArgs e)
            {

            }

            private void Transform_Click(object sender, System.EventArgs e)
            {
                  //Instantiate the proxy class
                  Trans x_Trans = new Trans();

                  // Call the method of the proxy class
                  x_file = x_Trans.xmlToxml(txtXML.Text,txtXSL.Text);

                  // To send the output to the container within the page
                  txtResult.Text = x_file;
                  }

            private void Reset_Click(object sender, System.EventArgs e)
            {
                  txtXML.Text =" ";
                  txtXSL.Text =" ";
                  ResultFile.Text   = "";
                  txtResult.Text = " ";
                  RadioButtonList1.SelectedIndex = -1;

            }

            private void RadioButtonList1_SelectedIndexChanged(object sender,
System.EventArgs e)
            {

            }

            private void Send_Click(object sender, System.EventArgs e)
            {
```

```
                    if (RadioButtonList1.SelectedItem.Value.Equals("File") &&
ResultFile.Text.EndsWith("xml"))
                    {
                          StreamWriter wr = new StreamWriter(ResultFile.Text);
                          wr.Write(txtResult.Text);
                          wr.Close();
                    }
                    else
if(RadioButtonList1.SelectedItem.Value.Equals("New_Browser_Page"))
                    {
                          // To send the output to the new Browser Window
                          ASCIIEncoding ae = new ASCIIEncoding();
                          byte[] byteXML = ae.GetBytes(txtResult.Text);

                          Response.ContentType="text/xml";

      Response.OutputStream.Write(byteXML,0,byteXML.Length);
                          Response.End();
                    }
                    else
if(RadioButtonList1.SelectedItem.Value.Equals("New_Windows_Container"))
                    {
                          // To send the output to a conatiner out in a new
page
                          string ResultDoc = txtResult.Text;
                          Response.Redirect("FormContainer.aspx?ResultDoc=" +
System.Web.HttpUtility.UrlEncode(ResultDoc));
                    }

              }
      }
}
```

File Name: References.cs
For: The Web Based Client for the XML Transformer Web Service

```
//------------------------------------------------------------------------------
---
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.3705.0
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------
---

//
// This source code was auto-generated by Microsoft.VSDesigner, Version
1.0.3705.0.
//
namespace WebClientApplication.TransService {
    using System.Diagnostics;
```

```
    using System.Xml.Serialization;
    using System;
    using System.Web.Services.Protocols;
    using System.ComponentModel;
    using System.Web.Services;


    /// <remarks/>
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Web.Services.WebServiceBindingAttribute(Name="TransSoap",
Namespace="http://localhost/project/XMLTrans/")]
    public class Trans : System.Web.Services.Protocols.SoapHttpClientProtocol
{

        /// <remarks/>
        public Trans() {
            this.Url = "http://localhost/project/XMLTrans/Trans.asmx";
        }

        /// <remarks/>

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://localhost/
project/XMLTrans/xmlToxml",
RequestNamespace="http://localhost/project/XMLTrans/",
ResponseNamespace="http://localhost/project/XMLTrans/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        public string xmlToxml(string xmlinput, string xslinput) {
            object[] results = this.Invoke("xmlToxml", new object[] {
                        xmlinput,
                        xslinput});
            return ((string)(results[0]));
        }

        /// <remarks/>
        public System.IAsyncResult BeginxmlToxml(string xmlinput, string
xslinput, System.AsyncCallback callback, object asyncState) {
            return this.BeginInvoke("xmlToxml", new object[] {
                        xmlinput,
                        xslinput}, callback, asyncState);
        }

        /// <remarks/>
        public string EndxmlToxml(System.IAsyncResult asyncResult) {
            object[] results = this.EndInvoke(asyncResult);
            return ((string)(results[0]));
        }
    }
}
```

File Name: SQLForm.aspx.cs
For: The XML-SQL Web Application

```
using System;
using System.Collections;
using System.Data;
using System.ComponentModel;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.IO;
// Required for the SQL Methods and functinality
using System.Data.SqlClient;
// Required fro the XML methods and functinality
using System.Xml;
// Required to ASCIIEncoding functinality
using System.Text;

namespace XML_SQLApplication
{
	public class XMLSQLForm : System.Web.UI.Page {
		protected System.Web.UI.WebControls.RadioButtonList
RadioButtonList;
		protected System.Web.UI.WebControls.Label FileNameLabel;
		protected System.Web.UI.WebControls.Label SendResultLabel;
		protected System.Web.UI.WebControls.Label XMLSQLDocumentLabel;
		protected System.Web.UI.WebControls.Button Send;
		protected System.Web.UI.WebControls.TextBox XMLSQLDoc;
		protected System.Web.UI.WebControls.TextBox ResultDoc;
		protected System.Web.UI.WebControls.Button Run;
		protected System.Web.UI.WebControls.Button Reset;
		protected System.Web.UI.WebControls.Label ResultLabel;
		protected System.Web.UI.WebControls.TextBox FileName;
		protected System.Web.UI.WebControls.Label Label1;

		private string DBServer;
		private string DBSecurity;
		private string Database;
		private string DBTimeout;

		private string ConnectionString;

		private string DataSetValue;
		private string ResultSetValue;
		private string SQLQueryElement;


		private void Page_Load(object sender, System.EventArgs e) {
			// Put user code to initialize the page here
		}

		#region Web Form Designer generated code
```

```
		override protected void OnInit(EventArgs e) {
			//
			// CODEGEN: This call is required by the ASP.NET Web Form
Designer.
			//
			InitializeComponent();
			base.OnInit(e);
		}

		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent() {
			this.XMLSQLDoc.TextChanged += new
System.EventHandler(this.XMLSQLDoc_TextChanged);
			this.ResultDoc.TextChanged += new
System.EventHandler(this.ResultDoc_TextChanged);
			this.Run.Click += new System.EventHandler(this.Run_Click);
			this.Reset.Click += new
System.EventHandler(this.Reset_Click);
			this.RadioButtonList.SelectedIndexChanged += new
System.EventHandler(this.RadioButtonList_SelectedIndexChanged);
			this.FileName.TextChanged += new
System.EventHandler(this.FileName_TextChanged);
			this.Send.Click += new
System.EventHandler(this.Send_Click);
			this.Load += new System.EventHandler(this.Page_Load);

		}
		#endregion

		private void Run_Click(object sender, System.EventArgs e) {

			//XML Writer send the data to a Memory Stream
			XmlTextWriter myWriter;
			Stream s = new MemoryStream();
			myWriter = new XmlTextWriter(s, null);
			myWriter.Formatting = Formatting.Indented;

			//Calling the method that will read the XML-SQL Document
Text Box
			//to find out wither it is a Document or a file
			XmlTextReader myReader = ReadXmlSqlTextBox();
			myReader.WhitespaceHandling =
WhitespaceHandling.Significant;

			Stack myStack = new Stack();

			// Reading and Writing Loop
			while(myReader.Read()) {
				switch(myReader.NodeType) {
					case XmlNodeType.Element:
						if(myReader.Name == "server"){
							myStack.Push(myReader.Name);
						}
						else if(myReader.Name == "security"){
```

```
                                                myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name == "database"){
                                                myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name == "timeout"){
                                                myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name=="DataSetElement")
{
                                                myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name =="ResultElement")
{
                                                myStack.Push(myReader.Name);
                                        }
                                        else if(myReader.Name=="SQLQuery") {
                                                myStack.Push(myReader.Name);
                                        }
                                        else if (myReader.Name == "SQLStatement")
{
                                                myStack.Push(myReader.Name);

        myWriter.WriteStartElement("SQLQueryResult");
                                                myWriter.WriteAttributes(myReader,
true);
                                        }
                                        else {
                                                myStack.Push(myReader.Name);

        myWriter.WriteStartElement(myReader.Name);
                                                myWriter.WriteAttributes(myReader,
true);
                                        }
                                        break;

                                case XmlNodeType.CDATA:
                                        myWriter.WriteCData(myReader.Value);
                                        break;

                                case XmlNodeType.EndElement:
                                        if ((myReader.Name== "DataSetElement") ||
(myReader.Name== "ResultElement") || (myReader.Name==
"SQLQuery")||(myReader.Name== "server")|| (myReader.Name== "security") ||
(myReader.Name== "database")||(myReader.Name== "timeout")) {
                                                myStack.Pop();
                                        }
                                        else {
                                                myWriter.WriteEndElement();
                                                myStack.Pop();
                                        }
                                        break;

                                case XmlNodeType.EntityReference:
                                        myWriter.WriteEntityRef(myReader.Name);
                                        break;
```

```
                              case XmlNodeType.ProcessingInstruction:
      myWriter.WriteProcessingInstruction(myReader.Name,myReader.Value);
                                    break;

                              case XmlNodeType.Text:
                                    // Call the method that will peek in top
of the stack
                                    // and extract the DB elements
                                    ExtractDBElements(myStack, myReader,
myWriter);
                                    break;

                              //case XmlNodeType.SignificantWhitespace:
                              //    myWriter.WriteString(myReader.Value);
                              //    break;

                              default:
                                    break;
                        }// End switch

                  }// End While Loop

                  myReader.Close();

                  // Writing the Result Document to the Result Text Box
                  myWriter.Flush();
                  s.Position = 0;
                  StreamReader sr = new StreamReader(s);
                  ResultDoc.Text = sr.ReadToEnd();

                  myWriter.Close();

            }

            private void ExtractDBElements(Stack myStack, XmlTextReader
myReader, XmlTextWriter myWriter){

                  if (myStack.Peek().Equals("server")) {
                        DBServer = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("security")) {
                        DBSecurity = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("database")) {
                        Database = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("timeout")) {
                        DBTimeout = myReader.Value;
                        // build the Connection String
                        buildConnectionString();
                  }
                  else if (myStack.Peek().Equals("DataSetElement")) {
                        DataSetValue = myReader.Value;
                  }
                  else if (myStack.Peek().Equals("ResultElement")) {
                        ResultSetValue = myReader.Value;
```

```
                    }
                    else if (myStack.Peek().Equals("SQLQuery")) {
                          SQLQueryElement = myReader.Value;

                          // Connect to Database and return the Result in XML
Format
                          string ResultXML = DatabaseConnection();

                          // Crate an XmlTextReader object Reader1 to Read the
string from DB
                          ASCIIEncoding AEnCod = new ASCIIEncoding();
                          byte[] XMLbyte = AEnCod.GetBytes(ResultXML);
                          MemoryStream myStr = new MemoryStream(XMLbyte);
                          XmlTextReader Reader1 = new XmlTextReader(myStr);

                          while(Reader1.Read()) {
                                // Reading and Write the Result in between the
SQLStatament Element/Tag
                                ReadingWritingDataSet(Reader1, myWriter,
DataSetValue);
                          }
                          Reader1.Close();
                          myStr.Close();
                    }// Close the if for the "SQLQuery" Element
                    else {
                          myWriter.WriteString(myReader.Value);
                    }
              }

              private void buildConnectionString() {

                    // Adding ";" to each string claus
                    string ServerStr = DBServer.Insert(DBServer.Length, ";");

                    string SecurityStr =
DBSecurity.Insert(DBSecurity.Length,";");

                    // Concatinating the 1st two strings the Server and the
Security
                    string tempStr1 =
ServerStr.Insert(ServerStr.Length,SecurityStr);

                    string TimeoutStr = DBTimeout.Insert(DBTimeout.Length,";");

                    // Concatinating/Adding the Timeout to the tempStr1
                    string tempStr2 =
tempStr1.Insert(tempStr1.Length,TimeoutStr);

                    //Building the Connection String. Note: No need for ";" for
the last claus of the string
                    ConnectionString = tempStr2.Insert(tempStr2.Length,
Database);

              }

              private string DatabaseConnection() {
```

```
                    // Conecting to the Database
                    string conStr = ConnectionString;

                    // Connecting to Database and Open the Connection
                    SqlConnection sqlConn = new SqlConnection(conStr);
                    sqlConn.Open();

                    // Running the Query
                    SqlDataAdapter da = new SqlDataAdapter(SQLQueryElement,
sqlConn);

                    // Passing the DataSet Element
                    DataSet ds = new DataSet(DataSetValue);

                    // Passing the Result Element
                    da.Fill(ds, ResultSetValue);

                    // Instainating a string builder to hold the XML Result in
to a string
                    StringBuilder BuildResult = new StringBuilder();
                    TextWriter sw = (TextWriter) new StringWriter(BuildResult,
null);

                    // Writing the Result in XML Format
                    ds.WriteXml(sw);
                    sqlConn.Close();

                    string ResultDoc = BuildResult.ToString();

                    return ResultDoc;
            }

            private void ReadingWritingDataSet(XmlTextReader Reader1,
XmlTextWriter myWriter, string DataSet) {

                    switch(Reader1.NodeType) {
                          case XmlNodeType.Element:
                                if (Reader1.Name == DataSet) {
                                      // Writing the Query statment as an
attrbute element
                                      // <sql:query> in the DataSet Element
                                      myWriter.WriteStartElement(Reader1.Name);

    myWriter.WriteAttributeString("sql:query",SQLQueryElement);
                                      // Writing the Database Name as a
attrbuite element
                                      // <from> in the DataSet Element

    myWriter.WriteAttributeString("source",Database);
                                }
                                else {
                                      myWriter.WriteStartElement(Reader1.Name);
                                      myWriter.WriteAttributes(Reader1, true);
                                }
                                break;

                          case XmlNodeType.CDATA:
```

```
                        myWriter.WriteCData(Reader1.Value);
                        break;

                  case XmlNodeType.EndElement:
                        myWriter.WriteEndElement();
                        break;

                  case XmlNodeType.EntityReference:
                        myWriter.WriteEntityRef(Reader1.Name);
                        break;

                  case XmlNodeType.ProcessingInstruction:

      myWriter.WriteProcessingInstruction(Reader1.Name, Reader1.Value);
                        break;

                  case XmlNodeType.Text:
                  case XmlNodeType.SignificantWhitespace:
                        myWriter.WriteString(Reader1.Value);
                        break;

                  default:
                        // write nothing to the result document
                        break;
            }
      }

      private void Send_Click(object sender, System.EventArgs e) {

            if
(RadioButtonList.SelectedItem.Value.Equals("New_Browser_Window")) {
                  ASCIIEncoding ae = new ASCIIEncoding();
                  Response.ContentType="text/xml";
                  byte[] byteXML = ae.GetBytes(ResultDoc.Text);

      Response.OutputStream.Write(byteXML,0,byteXML.Length);

                  Response.End();
            }
            else if (RadioButtonList.SelectedItem.Value.Equals("File"))
{
                  StreamWriter wr = new StreamWriter(FileName.Text);
                  wr.Write(ResultDoc.Text);
                  wr.Close();
            }

      }

      private void Reset_Click(object sender, System.EventArgs e) {

            XMLSQLDoc.Text = " ";
            ResultDoc.Text = " ";
            FileName.Text = " ";
            RadioButtonList.SelectedIndex = -1;

      }
```

```
            private XmlTextReader ReadXmlSqlTextBox() {
                  XmlTextReader Reader;

                  // Reading the Document/Data from File
                  if (XMLSQLDoc.Text.EndsWith("xml")) {
                        Reader = new XmlTextReader(XMLSQLDoc.Text);
                  }
                  else {
                        // Memory stream reading the XML Document/Data
                        byte[] byteXML =
Encoding.UTF8.GetBytes(XMLSQLDoc.Text);
                        MemoryStream XMLStream = new MemoryStream(byteXML);
                        Reader = new XmlTextReader(XMLStream);
                  }
                  return Reader;
            }

            private void XMLSQLDoc_TextChanged(object sender,
System.EventArgs e) {

            }

            private void ResultDoc_TextChanged(object sender,
System.EventArgs e) {

            }

            private void RadioButtonList_SelectedIndexChanged(object sender,
System.EventArgs e) {

            }

            private void FileName_TextChanged(object sender, System.EventArgs
e) {

            }
      }
}
```