Developing XML Web Services

Introduction
In this tutorial you will learn how to build XML Web Services in .Net.

Required Software
For this tutorial, you will need access to the following software applications:
Browser
VISUAL STUDIO

Prerequisites
Forest must be refactored to relocate the database from Presentation Tier to the Data Tier.

Reference
S Walther 'ASP.NET 4.0 Unleashed', SAMS

Breakdown of Tutorial
This tutorial consists of the following tasks:

- Exploring Web Service concepts and potential E-Commerce application.
- Developing the *Calculator* XML Web Service. Developing a client application that consumes this web service.
- Developing an XML Web Service that exposes one of the *Forest*'s music business functionalities. Examining the XML Web Service by launching it in a browser.

Refactor Forest

Download and follow instructions:	 Preparing to Implement Web Ser

Refactor Forest

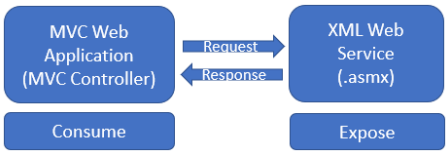
Exploring Web Service concepts
During the lecture, you were introduced to the components and protocols relating to XML Web services. We will now explore some practical examples of real-life Web services to help you to master these concepts.

	What does SOAP stand for? What functions do SOAP and HTTP perform within a Web service?
	What role does XML play in Web Services? What is WSDL?

	List below, 3 e-business scenarios for which XML Web services are ideally suited (both consuming and exposing)

Developing XML Web Services
We will start by building a basic web service to get an insight. Later, we will build a web service which exposes *Forest* music business functionality. Also, we will not be completely integrating these Web services into an application, which means that we will be dealing with raw XML formatted responses. In the next tutorial, we will be employing the Web service in a much more sophisticated and realistic way.

	Note that we will regularly mention the concept of a *Web service consumer* – What do we mean by the consumer of a web service?

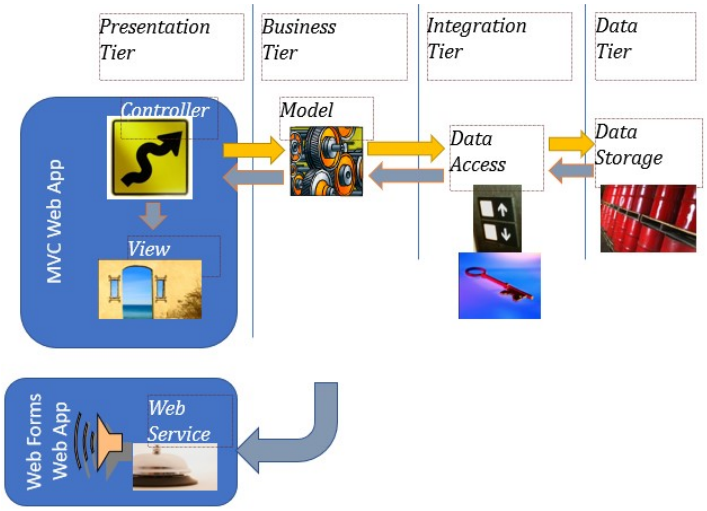
	Service Provider	To get a simple system that uses a .NET Web service up and running, there must be a service provider. In .NET, a service provider may be a **.asmx** service. Service needs to be hosted and for that we need a Web Application.	

	Service Consumer	Service will be called by a service consumer. In .NET, a service consumer will be a class in any of the different types of projects that is available in .NET. For example, a class in a Class Library, a controller in an MVC Web Application, a Web Form in a WebForm Application, a class in a Console Application, etc.	

The **asmx** file contains the class structure for the Web service. The asmx file also contains all the public functions and private procedures which allow the Web service to expose useful operations to the Web service consumer.

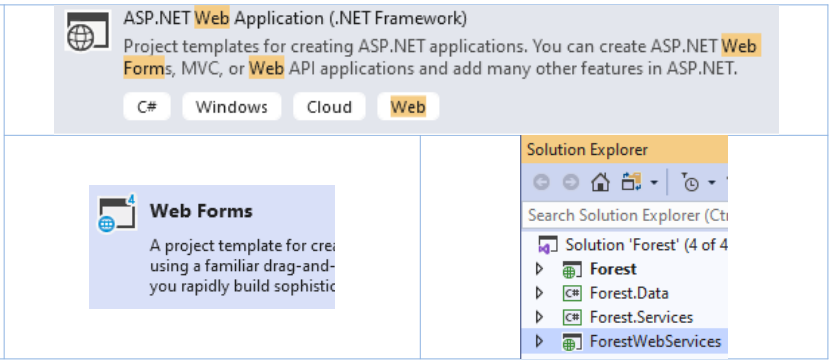
We are using C# to code our Web service. However, you could use any other .NET language which can execute under the Common Language Runtime. In fact, you can use C#, J#, C++, Jscript.NET Etc to build your Web service. In practice, there should be no discernible difference in performance between any of these languages. Nevertheless, for all purposes of this module we only support C#.

In summary, we **build and expose** the **Web service** (a *.asmx*). We host our web service in a Web Forms web Application.



Next lesson, we will build a class Library to consume a Service.

Creating a Web Forms Web Application for hosting Forest Web Services

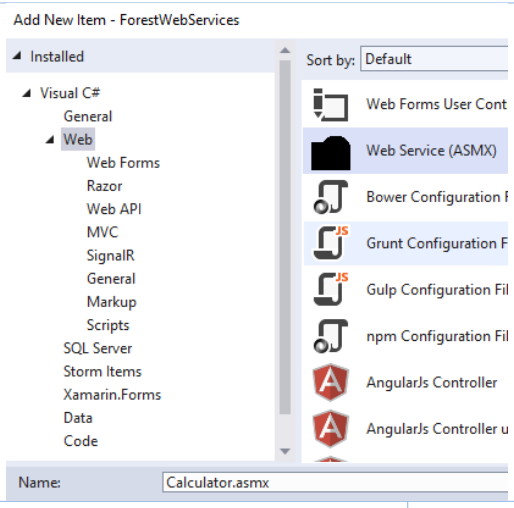
_ Launch Visual Studio _ Open Forest solution _ Add a new a Web Application project of type Web Forms to the solution _ Name the project 'ForestwebServices'	

Developing the *Calculator* XML Web Service

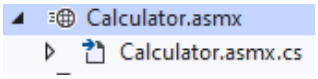
Object of this exercise is to create a new Web Service. *Calculator* Web Service would have methods for basic arithmetic functions. For example, *Add* method adds two numbers and returns the result.

_ Add a new item to the Website.

- Name: Calculator.asmx
- Type: Web Service (ASMX)
- Language: C#



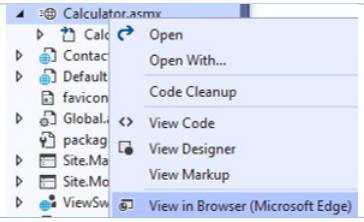
Note - Note that two files are created. One is *Calculator.asmx* and the other is *Calculator.asmx.cs*. The *.cs* file is a class. Note that Web Services do not have user interface and so all the work that we need to do, we will do in the .cs file (in the class).

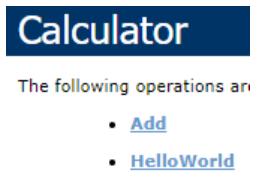


Working on *Calculator.asmx.cs*;
_ Add a new *WebMethod*

```
[WebMethod]
0 references
public double Add(double a, double b)
{
    return a + b;
}
```

To test your Web Service;
_ Right-Click on Calculator.asmx, and view in Browser.

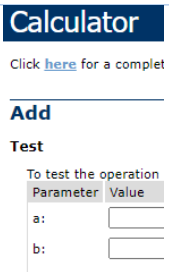




Learning checkpoints

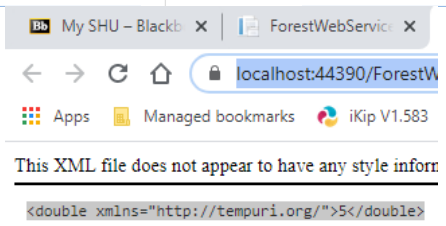
Click on *Add* Web Method and you should get the following HTML form and the various schemas for the Web Method messages;

What are the messaging protocols that this Web Service supports? What are the schemas for each of the messages?



Invoke *Add* Web Method for adding 2 and 3 together and you should get the following XML response;

What is the schema that this response conforms to?



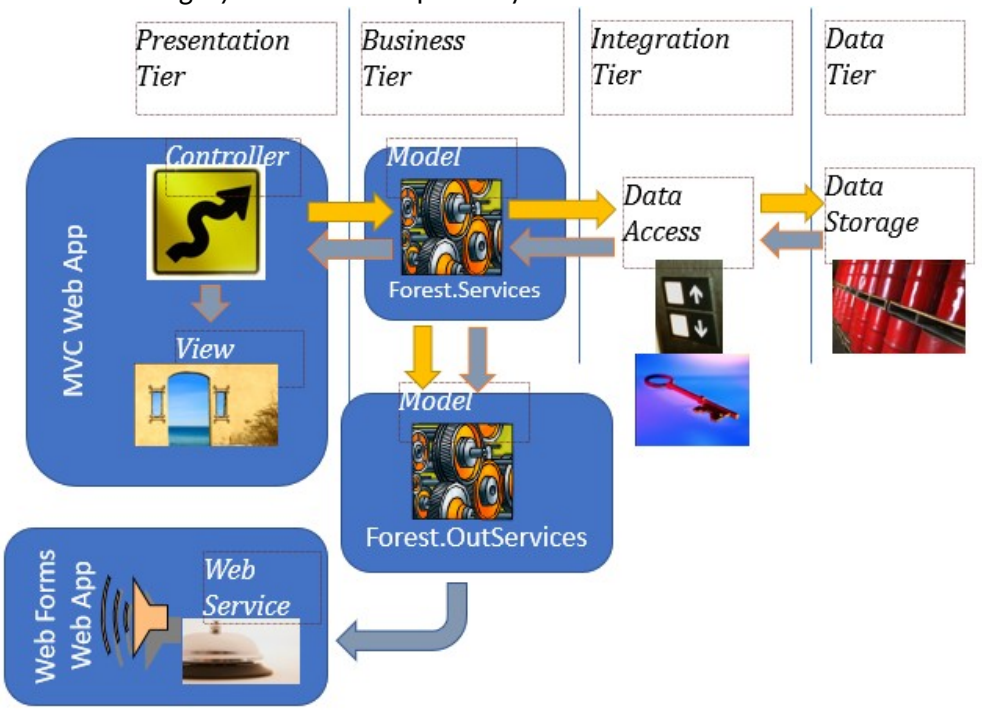
_ Extend the Calculator Web Service to include a Web method for subtraction

Developing XML Web Services that expose *Forest*'s business functionalities

It makes some sense to prepare special method for preparing the data that Forest serves to the other applications using web services. This may be for several reasons:

1. It is possible that *Forest* would want to expose using special data schema that does not necessarily look the same as those within *Forest*. Data should be specially prepared for the remote applications because response may require more comprehensive dataset, or Forest would simply not want to give its schema away.
2. XML Web Services serialise data into XML. Some of the classes that *Forest* works with may not necessarily serialise. Interface types do not serialise: *IList* and *ICollection* are types that *Forest* uses and these do not serialise. And alternative types such as *List* or *Array* should be used to facilitate serialisation.

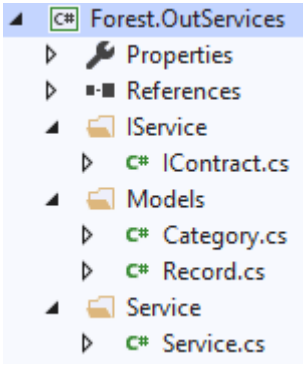
We prepare a business tier class library for preparing those methods that will be exposed as web Services. We call this class library '*Forest.OutServices*'. The object of this exercise is to expose a Web Service method for serving list of *Genre*. *Genre* and *Music* will be called *Category* and *Record* respectively.



Create and prepare the *Forest.OutServices* class library

Working in *Forest* solution ..
_ Create the *Forest.OutServices* class library (.NET Framework, C#)

Working in *Forest.OutServices* solution ..
_ Add references to Forest.Services and Forest.Data projects
_ Add Folders, Interface, and Classes



Implement the *IContract* interface and the *Service* class

Working in *Forest.OutServices* solution ..

_ Implement class *Record*

```
namespace Forest.OutServices.Models
{
    19 references
    public class Record
    {
        1 reference
        public Record() {}
        1 reference
        public Record(int id, string title, double price, string released)
        {
            ID = id; Title = title;
            Price = price; Released = released;
        }
        4 references
        public int ID { get; set; }
        4 references
        public string Title { get; set; }
        4 references
        public double Price { get; set; }
        4 references
        public string Released { get; set; }
    }
}
```

Working in *Forest.OutServices* solution ..
_ Implement class *Genre*

```
using Forest.OutServices.Models;
namespace Forest.OutServices.Models
{
    0 references
    public class Category
    {
        0 references
        public int ID { get; set; }
        0 references
        public string Name { get; set; }
        0 references
        public Record[] Records { get; set; }
    }
}
```

_Implement *IContract* interface

```
using Forest.OutServices.Models;
namespace Forest.OutServices.IService
{
    3 references
    public interface IContract
    { // GetMusicCategories() method is to return Array of Category
        0 references
        Category[] GetMusicCategories();
    }
}
```

_Implement *Service* class

```
using Forest.OutServices.IService;
using Forest.OutServices.Models;
using Forest.Data.Models.Domain;
namespace Forest.OutServices.Service
{
    1 reference
    public class Service : IContract
    {
        Forest.Services.IService.IMusicService musicService;
        Forest.Services.IService.IGenreService genreService;
        2 references
        public Service()
        {
            musicService = new Forest.Services.Service.MusicService();
            genreService = new Forest.Services.Service.GenreService();
        }
        // GetRecords() method is a private method that converts list of Music
        //  to Array of Recortd. This method is called by GetMusicCategories() method.
        1 reference
        Record[] GetRecords(IList<Music> musicList)...
        1 reference
        public Category[] GetMusicCategories()...
    }
}
```

Implement *GetRecords(IList<Music> musicList)* method.

```
Record[] GetRecords(IList<Music> musicList)
{   // This is a private method that will be called by GetMusicCategories() method.
    // This method converts IList<Music> to Record[].
    // To implement:
    Record[] list = new Record[musicList.Count];
    for (int i = 0; i < list.Length; i++)
    {
        list[i] = new Forest.OutServices.Models.Record
        {
            ID = musicList[i].ID,
            Title = musicList[i].Title,
            Price = musicList[i].Price,
            Released = musicList[i].DateReleased.ToString()
        };
    }
    return list;
}
```

_Implement *GetMusicCategories()* method.

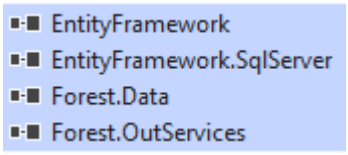
```
public Category[] GetMusicCategories()
{
    IList<Genre> genreList = genreService.GetGenres().ToList();
    Category[] array = new Category[genreList.Count];
    for (int i = 0; i < array.Length; i++)
    {
        array[i] = new Category
        {
            ID = genreList[i].ID,
            Name = genreList[i].Name,
            Records = GetRecords(genreList[i].Musics.ToList())
        };
    }
    return array;
}
```

_Build Forest.Outservices.

Implement the *GetMusicCategories* WebMethod

Working in *ForestWebServices* website ..

_Add References to assemblies and projects



_Copy and paste the *ForestContext* connectionString from '*Forest.Data/App.config*' to '*ForestwebServices/Web.config*'

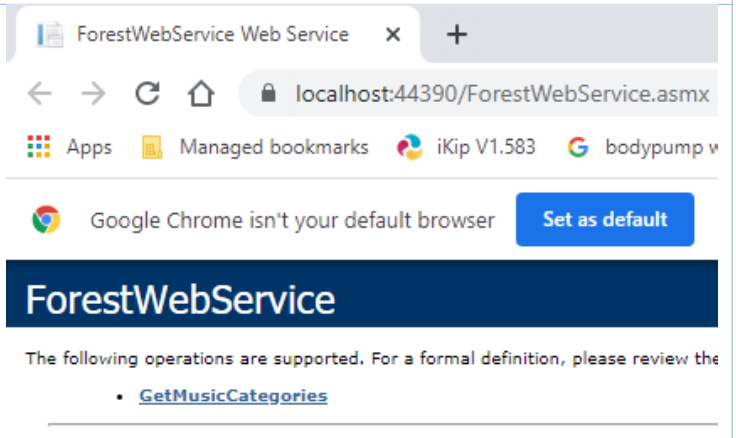
_ Add a new item to the Website.

- Name: ForestwebService.asmx
- Type: Web Service (ASMX)
- Language: C#

_Implement the *GetMusicCategories()* WebMethod

```
using Forest.OutServices.Models;
using Forest.OutServices.IService;
using Forest.OutServices.Service;
namespace ForestWebServices
{
    /// <summary>
    /// Summary description for ForestWebService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    1 reference
    public class ForestWebService : System.Web.Services.WebService
    {
        IContract service;
        0 references
        public ForestWebService()
        {
            service = new Service();
        }
        [WebMethod]
        0 references
        public Category[] GetMusicCategories()
        {
            return service.GetMusicCategories();
        }
    }
}
```

_Right click on ForestwebService.asmx and view in browser



_Invoke the method

What is the schema for the XML that is returned from the Web Service?