The objective of this lab is to become familiar with network programming using the XML Web Services API.

In this lab, you will build a web service that models in an extremely simplified fashion some of the functionality of an online book store such. Your service will in turn rely upon two commercial web services to provide its functionality: the web service APIs for amazon.com (Amazon Web Services) and google.com (Google Web APIs). In addition, you will build two clients to interact with your service – a console application for service management, and an ASP.NET Web Forms based application for providing a user-friendly front-end.

Preliminaries:

A. Compiling and Running the Lab4 Sample Applications

The discussion below refers to the starter files I have made available as a zipped-folder Lab4.zip on each of the development servers at the following location:

D:\VSDev\Public\vijay\Lab4.zip

Please expand this folder into the following location: D:\VSDev\<your user name>\Lab4.

This folder expands into a Visual Studio.NET solution, which contains one project linked into the solution: Client, and a subfolder containing four other projects (not linked into the solution): SumAndDifference, SubtractFive, StringReverser, and InteractionForm. The first three of these correspond to the web services I have been demonstrating in the lectures, while the last one provides the skeleton code for the the Web Forms based client.

1. Load the Lab4 solution into Visual Studio.NET.

Add four new projects into this solution. Three of these projects should be of type “Visual C# ASP.NET Web Service” and should correspond to the following URLs: http://localhost/VSDev/<your user name>/Lab4/WebServices/StringReverser (similarly, SubtractFive and SumAndDifference). The fourth project should be of type “Visual C# ASP.NET Web Application” and should correspond to the following URL: http://localhost/VSDev/<your user name>/Lab4/WebApplications/InteractionForm.

Save the solution and quit Visual Studio.NET.

2. Copy the contents of the CopyOutOfThisFolder subfolder provided in the Lab4 starter kit over the corresponding folders in your Lab4/WebServices and Lab4/WebApplications folders. To take an example, the contents of the CopyOutOfThisFolder\StringReverser folder should replace the (default) contents of WebServices\StringReverser.

Reload the Lab4 solution into Visual Studio.NET and rebuild. Other than the Client project, the remaining four projects should build without any errors.

3. Add two web references to the Client project: one to the StringReverser web service and the other to the SubtractFive web service. In the browser-like window that pops up when you try to add a web reference, type the URL corresponding to the WSDL for the two services (for example: http://localhost/VSDev/<your user name>/Lab4/WebServices/StringReverser/StringReverser.asmx?WSDL)

Make sure that the references are added with the names StringReverser and SubtractFive.
Rebuild the Client project, which should now build without errors. You should be able to run the Client project and verify that it produces the expected results.

B. Registering to use the Amazon.com and Google.com Web Services

You will need to register with Amazon.com and Google.com to make invocations against their web services. Registration is free and once you register, you receive a unique ID, which should be presented along with every service request you make.

1. The registration page for Amazon.com is accessible at the following URL:
   http://www.amazon.com/gp/aws/landing.html

   Follow the instructions to download the SDK (you need this primarily for the documentation about how to use the service), and to obtain the unique ID (referred to as the developer tag).

   Amazon.com’s web service offers about 30 operations. You will be using only one of these: **AsinSearchRequest**, which takes as input an ASIN (a unique number identifying each of Amazon’s products), and produces a ProductInfo structure as response. For this lab, you will be working with ASINs that correspond to the ISBN designations of books (I have supplied a list for your experimentation). For such ASINs, the Amazon.com web service returns information about the book’s title, its authors, the publisher, the list and amazon.com price, etc. You should look at the documentation to fully understand the fields of the ProductInfo structure. Additionally, when you invoke the service, make sure that you specify the following options in your request (in addition to the ASIN and your unique ID): **mode=“books”, tag=“webservices-20”, type=“lite”**

   The following URL returns the WSDL description of the Amazon.com web service (in Visual Studio.NET, this is the URL that should be provided when you are setting up a web reference to build the proxy):

   The proxy class that gets created is called **AmazonSearchService** (AsinSearchRequest is a method in this class).

2. The registration page for Google.com is accessible at the following URL:
   http://api.google.com

   As before, follow the instructions to download the SDK (again, you need this primarily for the documentation, which is also available from the above site) and to register. Unlike the Amazon.com site, which has you fill out a simple form to receive the ID, the Google site requires you to verify your e-mail address (by clicking on a link contained in a message sent to that address).

   Google.com’s web service offers 3 requests, of which you will be using one, **doGoogleSearch**. This basically corresponds to the query interface you should all be familiar with: it expects a query string as input (along with some other options) and responds with the search results. We will only use the URL of the highest-ranked document returned as the search result. You can control the number of results that are returned by setting an appropriate request option. Again, take a look at the documentation to understand the different fields in the request and response structures.

   The following URL returns the WSDL description of the Google.com web service (you will supply this in the web reference window):
   http://api.google.com/GoogleSearch.wsdl
What You Need To Do:

1. Create a project, MyBookService, of type “Visual C# ASP.NET Web Service” under the Lab 4 solution. This service should correspond to the following URL: http://localhost/VSDev/\<your user name>/Lab4/WebServices/MyBookService. Rename the .asmx file created by default to something more reasonable (make sure that you change the namespace, main class names in this file as well).

This service should provide the following operations:

[Management interface]

void PopulateDatabase( string[] isbn_list )
Should take an ISBN list as input and populate the service’s state with records corresponding to these ISBNs. Each record should contain the following fields: isbn (string), title (string), authors (string[]), publisher (string), price (string). The ISBN list I have provided in the root Lab4 folder corresponds to real books from the bestseller list at Amazon.com. To provide the functionality required by this operator, your service will need to invoke the AsinSearchRequest operator on Amazon’s web service, and extract the desired fields from the return structure.

Note that the Amazon.com web service does not allow more than a certain number of requests every time unit, so don’t be alarmed if your program seems to be running slowly. That said, it should not take more than a few seconds for each request to complete.

Another point: as with the XML-RPC application, where different requests to the web service could potentially run in different processes, you should not assume that multiple requests to the web service run in the context of the same process. This means that your service must explicitly save/restore any state that needs to be available across requests (the records are an example of such state). I would suggest that you use the appstate folder configuration that I specified in Lab 2. Make sure that the folder’s permissions are set right so as to allow the ASP.NET process to write to it.

string[] ListAllISBNs( )
Lists the ISBNs of the records known to the service.

void UpdatePrice( string isbn, string newPrice )
Modify the price field of the record to reflect the new price.

[User interface]

string[] SearchISBNs( string keyword_query, string price_min, string price_max )
Should return an array of ISBNs that correspond to the keyword and price criteria.

Keyword queries are of the form: “word1 AND word2” or “word1 OR word2”. It is possible to have an empty word2, in which case the keyword query is interpreted as just word1 alone. It is possible to use a “*” in word1 position. It is an error to have an empty word1 and a non-empty word2.

Price queries are specified in terms of min- and max- prices. The string arguments should be convertible to integers without raising exceptions. It is an error to have a price_min larger than price_max.

The query returns the ISBNs of those records that satisfy both the keyword and price queries. A keyword query of “*” is deemed to match all of the records. A record satisfies a keyword query if the keyword words appear in any of the title, authors, or publisher fields.

string GetRecords( string[] isbn_list )
Returns the fields of the requested records in a human-readable plain-text form. For example:

ISBN: <isbn>
TITLE: <title> …

For each ISBN, the following three fields need to be dynamically generated (in response to the request): a URL pointing to an image of the book, the current Amazon.com price for the same item, and a URL pointing to the Amazon.com page describing the book. The first two of these can be obtained by
invoking the AsinSearchRequest operation on the Amazon.com web service (and picking up fields that were ignored in the PopulateDatabase call). The third requires an invocation on the Google.com web service, say passing it the following query string: “<name of book>” “Editorial Reviews” site:www.amazon.com, and picking up the URL of the first search result.

```csharp
string GetRecordsHTML( string[] isbn_list )
```

Provides the same functionality as GetRecords, except for the fact that the string that is returned in response is formatted HTML. One possibility is to create a table so that the image of the book shows up on the left, the book details in the center, and the prices at MyBookService and Amazon.com to the right.

Note that to access the Amazon.com and Google.com web services, you will need to add the corresponding web references to obtain the proxy code that your application can instantiate and use.

2. Create a console application, MyBookServiceAdmin, to test the administrative interface of the web service. Your program should be able to populate the database with one or more ISBNs, list all known ISBNs, and change the prices of individual books. You may find it helpful to also exercise the user interface through this program, namely the SearchISBNs and GetRecords operations.

3. Extend the InteractionForm functionality so that it invokes SearchISBNs and GetRecordsHTML operations on the service to render the results appropriately to the user. The user accesses the web form using the following URL:

   http://localhost/VSDev/<your user name>/Lab4/WebApplications/InteractionForm/Start.aspx

4. Handle errors and exceptions as gracefully as you can (e.g., by catching the appropriate exception and printing out an error message or informing the user about the error via appropriately generated HTML.

Submit the lab folder information and writeup using the web-based submission form on the development server where you have your account. If you follow the guidelines above, the folder should just be: D:\VSDev\<your user name>\Lab4.