ASP.NET Web Services and Web Clients

Web Services Overview

- The World Wide Web has opened up the possibility of large-scale distributed computing
- Web Applications only allow interaction between a client browser and web server hosting a web page
- Web Services create web-based applications that interact with other apps on other computers
  - A web application is intended for viewing by a person using a browser
  - Web Service: a program with which other programs can interact without any user interface
  - Web Client: a program that consumes (uses, interacts with) a web service
    - Could be a Web Form, a Windows Form, or even a command line application
    - The web client usually has some sort of user interface

Some Example of Web Services

- There are lots of them out there
- http://www.xmethods.net has quite a few
- Microsoft's TerraService
  - Provides a programmatic interface to a massive database of geographic data
  - http://terraservice.net
- When you build a web client with Visual Studio, the "Add Web Reference" Browser tool can be used to find online services
  - UDDI (Universal Description Discovery Integration) Directories

ASP.NET Web Services

- Before ASP.NET, distributed computing was highly dependent on OS and language
- ASP.NET web services and clients are entirely independent of either
  - Could have a web service written in VB running on Windows 2000 consumed by a web client written in C++ running on a UNIX box
- What is needed?
  - That both client and server use industry standard protocols
    - HTTP – protocol used by the web
    - SOAP – Simple Object Access Protocol: a lightweight object-oriented communication protocol based on XML
    - XML – the language of SOAP

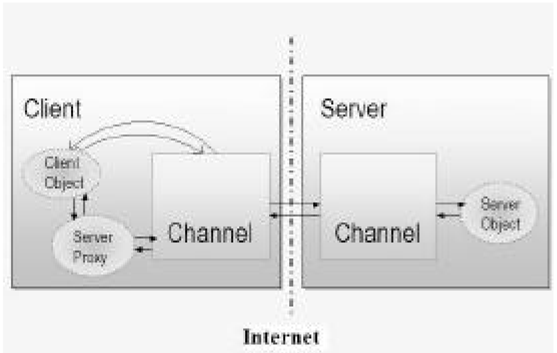
How Web Services Work

- A web service is simply a function or method call over the internet
  - Clients call exposed methods of the web service using standard internet protocols
  - Both client and server must be connected to the internet
  - Data format used to communicate is usually SOAP
    - Self-describing text-based XML documents
  - Systems at both ends of the connection are loosely coupled (OS, language, object model, etc. are not important)
    - Only requirement is that both server & client be able to send & receive messages that conform to the proper protocol standard

Sequence of Events

- Client makes a call to the web service
  - It appears as though it's talking directly to the web service over the internet
  - But the actual call is being made to a "proxy class" local to the client
    - Proxy is a substitute or stand-in for the actual code to be called
    - An object that provides a local representation of a remote service
    - It's really a DLL that handles all the complexities of sending requests over internet and getting responses back
    - " marshalls" the call to exposed methods across the internet
    - Proxy class object must be created by the client app
      - Done by Visual Studio when you create a "web reference"
      - Actually it's done by the Wsdl.exe (Web Services Description Language) utility program

Web Service/Client Interaction



Creating a Web Service w/ Visual Studio

- Easy with Visual Studio (IIS must be installed)
  - "File" | "New" | "Project"
    - "Project Type": C#
    - "Template": ASP.NET Web Service
    - "Location": <u>http://localhost/WebServiceName</u>
      - Project directory will be put in the home (Inetpub/wwwroot) directory of your IIS server
  - Creates .asmx and .asmx.cs files which contain skeleton code for the web service
  - .asmx file will contain the implementation of the web service
    - Note the "WEB SERVICE EXAMPLE HelloWorld()" method that has been commented out
    - Just add the methods you want the service to expose at that place in the .asmx .cs file

Example Web Service: ConvertTemperature

- Has temperature conversion methods ctf() and ftc():

```
[WebMethod (Description="Converts a Centigrade temperature to Fahrenheit")]
public decimal ctf(decimal ctemp)
{   return ((9M/5M)*ctemp + 32M); }
[WebMethod (Description="Converts a Fahrenheit temperature to Centigrade")]
public decimal ftc(decimal ftemp)
{   return (5M/9M)*(ftemp - 32M); }
```

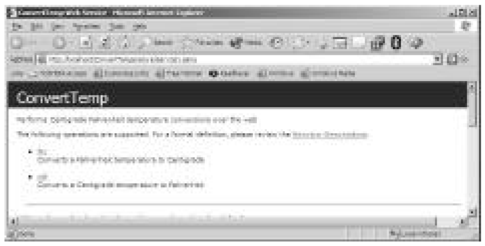
  - Note use of [WebMethod] attribute
    - Specifies that these methods are available to be used by web clients
    - Description will appear when service is tested in a browser
- Added to top of file: a [WebService] attribute

```
[WebService (Name="ConvertTemp", Description = "Performs Centigrade Fahrenheit
   temperature conversions over the web")]
```

  - "Name" and "Description" will appear in the HTML page generated when user calls up the service in a browser
    - "Name" determines name of Proxy class created by client

Running and Testing the Web Service

- Run the Web Service from Visual Studio just as for any other application
  - "Debug" | "Start without debugging"
  - Brings up the following web page in your browser:



ConvertTemp

Performs Centigrade Fahrenheit temperature conversions over the web

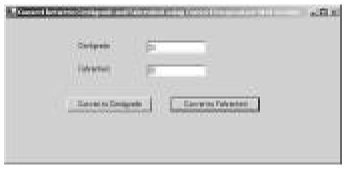
The following operations are supported. For a formal definition, please review the Service Description.

- ftc
  Converts a Fahrenheit temperature to Centigrade
- ctf
  Converts a Centigrade temperature to Fahrenheit

  - Clicking on ftc or ctf allows you to test the service's methods

Creating a Web Client for the Service

- Can use Visual Studio to build a Windows Form or Web Form application to use the Web Service
- Example "ConvertTempClient"
  - A Windows Form app



  - User enters Fahrenheit or Centigrade temperature in a textbox
  - Presses appropriate button
  - Other textbox will contain the converted temperature

Using VS to Create a Web Client

- Start a Windows Application project as usual
- Drag the controls over to the form and rename them as usual
- Add a <u>Web Reference</u>:
  - In Solution Explorer, right click on references
  - Click on "Add Web Reference"
    - "Add Reference Browser" page comes up
    - In Visual Studio 2003, click on "Browse to Web services on the local machine" and navigate to the web service
    - In Visual Studio 2002 you must type in the URL
      - <u>http://localhost/ConvertTemperature/Service1.asmx</u>
        - » If the service were on some other server, you'd specify its URL
    - Click "Add Reference" button
  - A new "Web References" folder also was created
  - Also notice in Class View that under {} localhost, a ConvertTemp class has been added
    - This is the proxy and contains the local representations of the ftc and ctf methods

Web Client Creation: Coding

- Double click the Convert to Centigrade button and add the following button click event handler code

```
localhost.ConvertTemp obj = new localhost.ConvertTemp();
string fstr = textBoxFahr.Text;
decimal ftemp = decimal.Parse(fstr);
decimal ctemp = obj.ftc(ftemp);
textBoxCent.Text = ctemp.ToString();
```

- Double click the Convert to Fahrenheit button and add the following button click event handler code

```
localhost.ConvertTemp obj = new localhost.ConvertTemp();
string cstr = textBoxCent.Text;
decimal ctemp = decimal.Parse(cstr);
decimal ftemp = obj.ctf(ctemp);
textBoxFahr.Text = ftemp.ToString();
```

- When you run the program, it will use the web service to perform the temperature conversions

Existing Web Services

- www.xmethods.net lists many of them
- Example: Zip Code Distance calculator from imacination.com web services
  - User provides zip codes of two cities and the service computes the distance between the cities
  - Computes other things as well
- We can use these or any other web services in our own Applications

A Zip Code Distance Client

- Creating a Web Client to use the "Zip Code Distance" web service from imacination.com
  - Use Visual Studio to create a new C# Windows Application (ZipDistance)
  - Add a web reference:
    - In Solution Explorer right click on ZipDistance and choose "Add Web Reference"
    - Type http://webservices.imacination.com/distance/ in the Address Field
    - Scroll down to the WSDL link and click on it
      - The wsdl file will appear in the left hand window
      - This is the "contract" describing the methods the server provides
      - It's generated automatically by Visual Studio Designer
    - Click the "Add Reference" button
      - This adds a new class to the project:
        - » ZipDistance.com.imacination.WebServices.DistanceService

ZipDistance User interface

- Drag over the following from the tool box:
  - Two text boxes (textBoxZip1, textBoxZip2)
  - Two label controls to label the text boxes
    - "First City Zip code"
    - "Second City Zip Code"
  - A label control to hold the computed distance (labelDistance)
  - A "Calculate Distance" button (buttonCalc)
- Add a button click event handler to the button

Coding the ZipDistance application

- Add code to the button's click event handler to:
  - retrieve the the zip codes entered into the two textboxes by the user
  - call the web service's GetDistance(string, string) method
  - set the labelDistance label control's Text property to the result (converted to a string):

```
private void buttonTemp_Click(object sender, System.EventArgs e)
{
   TempZipcodeClient.net.xmethods.www.TemperatureService obj =
          new TempZipcodeClient.net.xmethods.www.TemperatureService();
   labelTemp.Text = "Temperature is " +
          obj.getTemp(textBoxZip.Text).ToString();
}
```

- When you run it, you're using the remote web service