Module 4: Working with Web Services

In this section you will learn about:

- Convert a CFC into a Web Service
- Invoke an External Web Service

"Web Services" is the newest technology that "soon will allow refrigerators to stock themselves!" It seems that no matter what the newest technology is, it will always allow appliances to connect to the Internet. Actually, Web Services might just be the technology that allows that to happen (assuming, of course, that some grocery store will create such a web service and, that my refrigerator has the ability to connect to the Internet!)

A Web Service can be just about anything that a developer can think up! Defining one can be difficult in the same way that defining a "function" can be difficult. "Web Service" defines more about the structure than it describes what the Service actually does.

A Web Service is an application that is published on the Internet and is exposed so that others may use them. They use a set of open standards to allow for each access (including SOAP, HTTP, WSDL, UDDI and others.) These terms are defined below. In a sense, even the delivery of an HTML page is a web service. A browser uses HTTP to make a request of a server and the resulting code is returned with HTTP.

Examples

Let's consider some examples of Web Services that are currently available.

- **Real-time Baseball Statistics** – Submit a player's name or a team name and receive near real-time statistics
- **Zip Code +4 lookup** – Submit a valid US address and receive the full nine-digit zip code
- **Open Directory Project** – Submit search terms and receive a set of resulting websites that match
- **Spell checker** – Submit a piece of text and check it against one of several dictionaries
- **Weather** – Submit location information (city, zip code, etc) and receive local weather forecast

- **Stock Quote –** Submit a stock symbol and receive the current price and trading information

Is a Web Service the only way to get the above information? Obviously, no! It is possible to perform these very functions by building a private proprietary system, by using WDDX or another system .

So, why are Web Services so exciting? They provide a set of rules and standards that allow many technologies to use Web Services created in other technologies. For example, a Web Service that you create with ColdFusion may be consumed by a Microsoft .NET application and vice versa.

Producers vs. Consumers

In shear numbers, there are bound to be more consumers of Web Services than producers. For example, Google publishes a Web Service that makes its search results available on any site. This service is used by thousands of sites. Do the owners of those sites need to understand the inner programming involved in Google's Web Service? Absolutely not! If they did, certainly fewer sites would be using the service, not to mention the fact that Google would not want to publish something that would give a detailed look at their search algorithms or other proprietary information.

Consumers of Web Services need merely to understand:

1. **Available functions/methods –** What can the Service do? Does it have multiple functions? How are they different?

2. **Incoming Parameters –** What required and optional parameters does a function allow?

3. **Output –** What kind of output is expected?

Definitions

- **HTTP** – Hyper Text Transfer Protocol (HTTP) is used to transfer information to and from the Web Service. This is important because it is a well-established and existing standard.

- **XML** – SOAP, WSDL and other components of Web Services are either XML languages or use of XML is key to their operation.

- **SOAP** – Simple Object Access Protocol (SOAP) is a protocol that is involved in the translation of information as it is transferred to and from the Web Service. It is an XML language – one written using the XML specifications.

- **WSDL** – Web Services Description Language (WSDL) is an XML language that describes all available functions in the Service and also describes the parameters and output.
- **UDDI** – Universal Description, Discovery and Integration (UDDI) is a standard that originally proposed by Microsoft and IBM that allows for easy creation of directories of Web Services.

Without standards like HTTP, SOAP and WSDL, each application would have to build its own means to transfer and define information. Web Services provide a common structure that allow

Creating WSDL Documents

ColdFusion will automatically create a WSDL based on the information in your Web Service. There is no need to write your own, but a quick look at a WSDL is still a good idea. The standards for WSDL are available at: http://www.w3.org/TR/wsdl.

Demo: Converting CFCs into Web Services

Step 1 – Convert the existing CFC into a Web Service

Believe it or not, by simply adding the access attribute to a function in a CFC, it can be exposed as a Web Service.

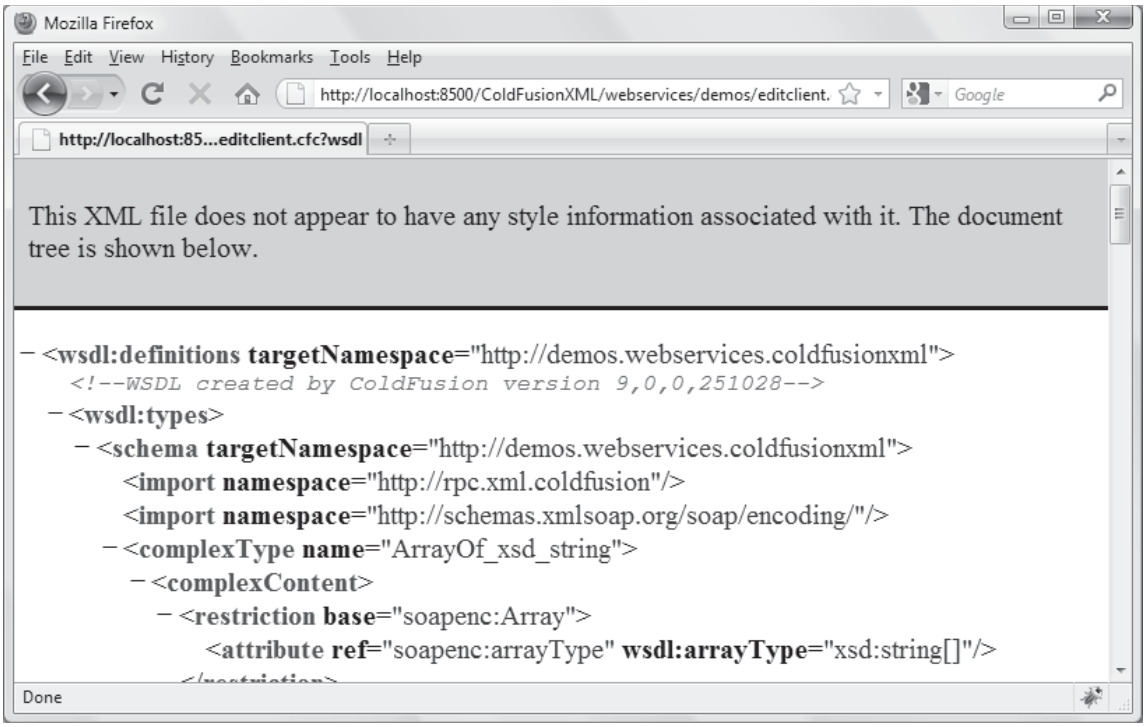
```
<cffunction name="getClients" returntype="query" access="remote">
```

That is not entirely true, there are some other requirements. For example, the returntype attribute must be set (where on a non-exposed CFC it is optional).

Step 2 - Examine a local WSDL file from the demo

ColdFusion will automatically create the WSDL file needed to access this Web Service. You may view it directly in the browser by adding "?wsdl" to the end of its URL:

http://localhost:8500/ColdFusionXML/webservices/demos/editclient.cfc?wsdl



Step 3 - Invoking a ColdFusion Web Service

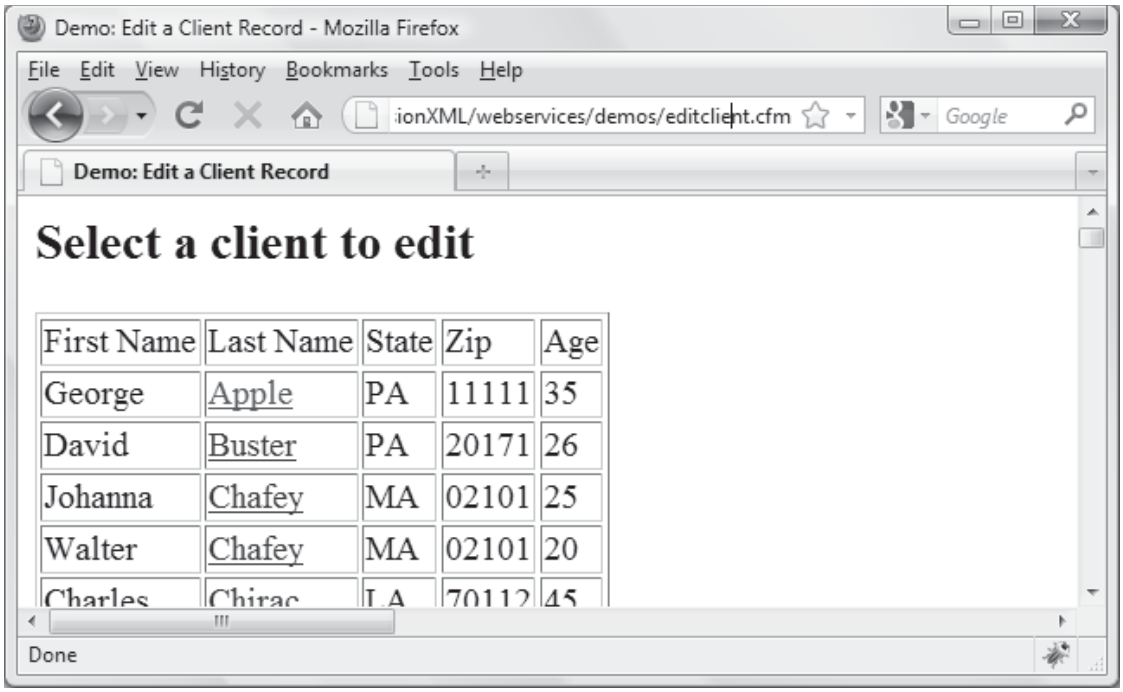
The <cfinvoke> tag calls the Web Service using the "webservice" attribute.

```
<cfinvoke method="getClients" returnvariable="clientrecords"
webservice="#REQUEST.prefix#webservices/demos/editclient2.cfc?wsdl" />
```

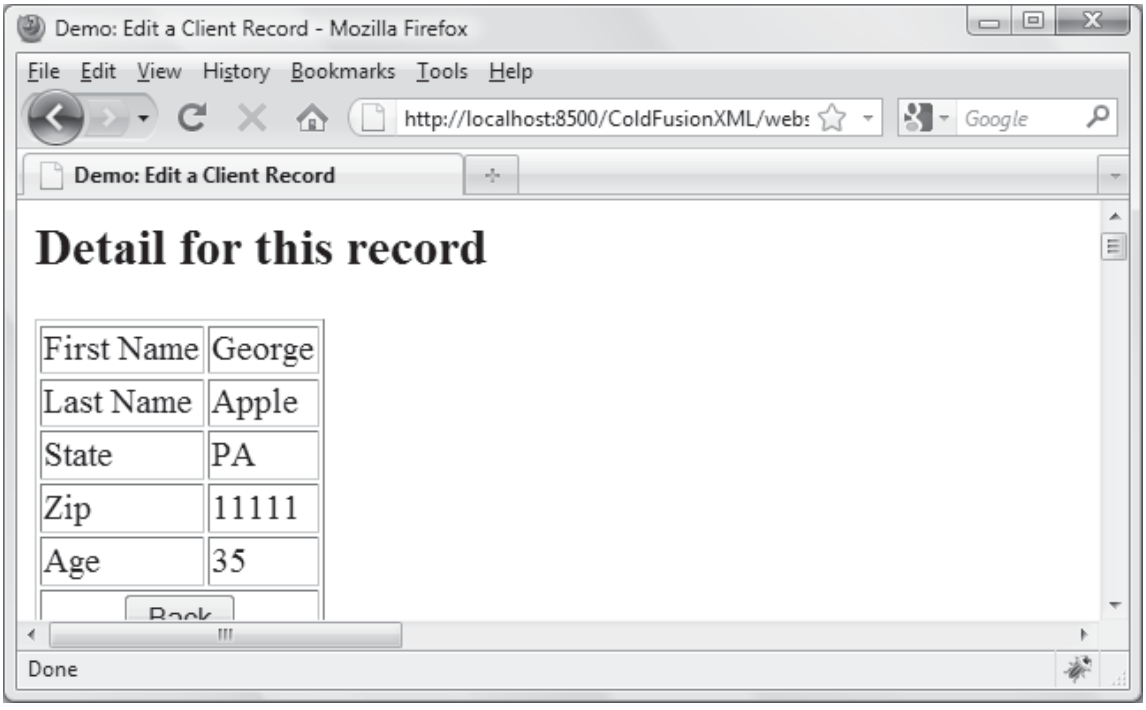
Deeper Examination

This demo is similar to one from the CFC section. It is saved as **webservices/demos/editclient.cfm.**

It has just two phases (where the other had three). Generally a Web Service would not allow for editing of our database (as the CFC did). It will provide information to the user. In "Phase 1," a list of clients is visible:



In Phase 2, the "detail" is visible:



Deeper Examination: Passing an argument to a Web Service

Just like with a CFC, an argument can be passed with the <cfinvokeargument> tag. There are, however, some additional restrictions. The type of the value passed must

match the expected type of the Web Service. Since ColdFusion does not strictly type values, the "clientid" variable must be converted using the Val() function into a numeric value.

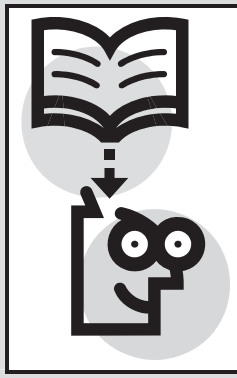
```
<cfinvoke method="getOneClient" returnvariable="onerecord"
webservice="#REQUEST.prefix#/webservices/demos/editclient.cfc?wsdl">
   <cfinvokeargument name="clientid" value="#Val(clientid)#" />
</cfinvoke>
```

SOAP

SOAP is similar in structure to WDDX in the sense that it takes data that is in one format, converts it into XML and allows it to be transferred using another protocol (like HTTP) to another location. Once this SOAP-encoded data is received, it may be converted into a consumable form. The translation to and from the server is done using the standard HTTP protocol.

A note about security

Exposing business logic in the form of Web Services can certainly be risky. You will want to be sure that you take appropriate precautions. For example, should you require registration or will your Web Service be available to anyone. If data that is being transferred to and from the server is sensitive information (such as passwords, credit card information, etc) consider encrypting it.

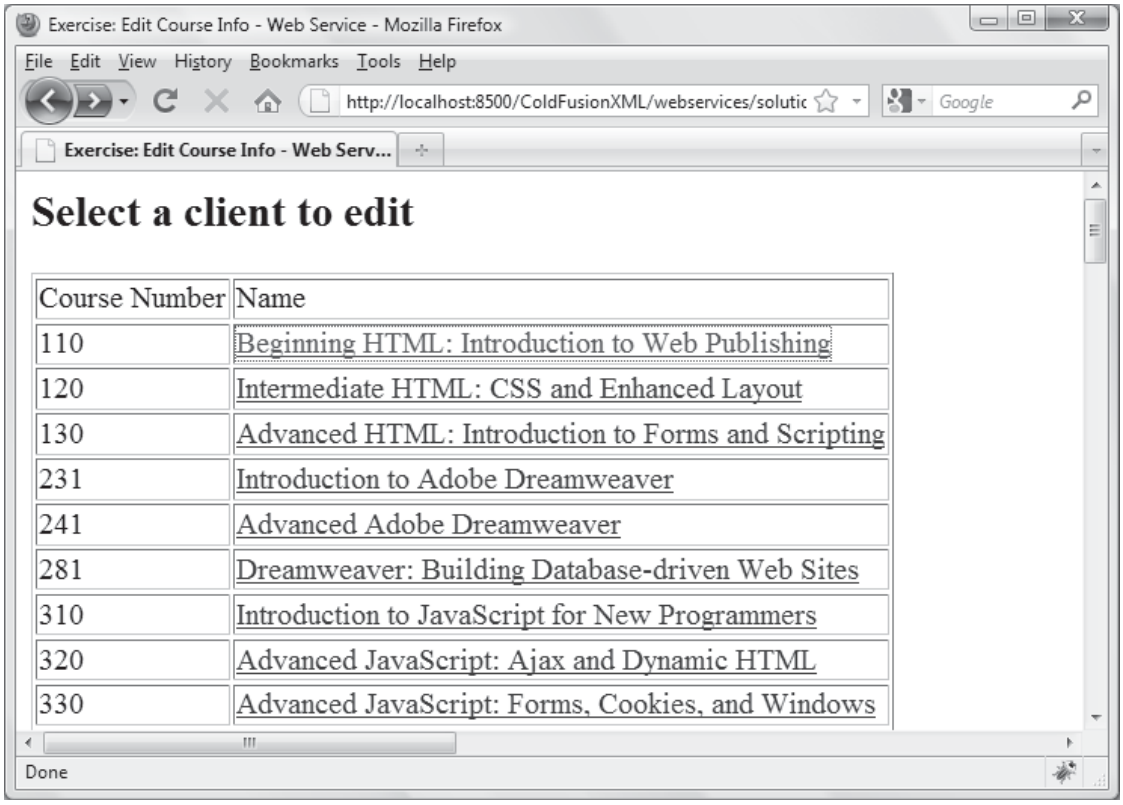


No hyphens in the names

You may have noticed that there are no hyphens in the names of the Web Service or the folders that they are located in. This is deliberate. Use of hyphens causes an error in Apache Axis.

Exercise 10: Convert Existing CFC into Web Service

The functions in the CFC will be exposed as Web Services and the calling page will invoke the Web Services.



For this exercise, complete the following:

1. Open the CFC page which is saved as **webservices/exercises/coursesWStemp.cfc**. Add the following attribute to the <cffunction> tags.

```
access="remote"
```

2. Open the calling page which is saved as **webservices/exercises/coursesWStemp.cfm** and you will find the following code:

```
<html>
<head>
   <title>Exercise: Edit Course Info - Web Service</title>
</head>
<body>
<cfif (NOT isDefined("seename") AND NOT isDefined("updatenow"))>
<!--- Phase 1 - list all courses  --->
```

```
   <!---
   Call the "getCourses" method of the coursesWStemp web service
   You will pass no arguments
   --->

   <h2>Select a client to edit</h2>
   <table border="1">
      <tr>
         <td>Course Number</td>
         <td>Name</td>
      </tr>
   <cfoutput query="clientrecords">
      <tr>
         <td>#coursenum#</td>
         <td><a
href="?seename=true&coursenum=#coursenum#">#name#</a></td>
      </tr>
   </cfoutput>
   </table>

<!--- Phase 2 - show current values for a course --->
<cfelseif isDefined("seename") AND isDefined("coursenum")>
   <!---
   Call the "getOneCourse" method of the coursesWStemp web service
   You will pass one argument - the "coursenum"
   coursenum must be passed as a numeric value
   --->
   <h2>View Details</h2>
   <cfoutput query="onerecord">
   <form method="post" action="<cfoutput>#cgi.script_name#</cfoutput>">
   <table border="1">
      <tr>
         <td>Course Number</td>
         <td>#coursenum#</td>
      </tr>
      <tr>
         <td>Name</td>
         <td>#name#</td>
      </tr>
      <tr>
         <td>Description</td>
         <td>#description#</td>
      </tr>
      <tr>
         <td align="center" colspan="5">
            <form>
            <input type="submit" value="Back">
            </form>
         </td>
      </tr>
      </cfoutput>
   </table>
</cfif>
</body>
</html>
```

3. Add two <cfinvoke> tags to **webservices/exercises/coursesWStemp.cfm**. The first will call the "getCourses" method of the coursesWStemp web service. You will pass no arguments. The second will call the "getOneCourse" method of the coursesWStemp web service. You will pass one argument - the "coursenum" which must be passed as a numeric value. Remember that you must add the attribute "**returntype="query"**"

Challenge

- Convert another existing CFC into a Web Service

A Possible Solution to Exercise 10

The dynamically generated WSDL file can be viewed in a browser at http://localhost:8500/ColdFusionXML/webservices/solutions/coursesWSdone.cfc?wsdl.

As saved in **webservices/solutions/coursesWSdone.cfc**:

```
<cfcomponent>
   <cffunction name="getCourses" access="remote" returntype="query"
output="No">
      <cfquery name="therecords" datasource="courses">
         SELECT * FROM courses ORDER BY coursenum
      </cfquery>
      <cfreturn therecords>
   </cffunction>
   <cffunction name="getOneCourse" access="remote" returntype="query"
output="No">
      <cfargument name="coursenum">
      <cfquery name="therecords" datasource="courses">
         SELECT * FROM courses
         WHERE coursenum = #arguments.coursenum#
         ORDER BY coursenum
      </cfquery>
      <cfreturn therecords>
   </cffunction>
</cfcomponent>
```

As contained in **webservices/solutions/coursesWSdone.cfm**:

```
<html>
<head>
   <title>Exercise: Using ColdFusion Components</title>
</head>
<body style="font-family:Arial">
<cfif NOT isDefined("coursenum")>
   <cfinvoke method="getCourses" returnvariable="clientrecords"
webservice="#REQUEST.prefix#webservices/solutions/coursesWSdone.cfc?wsdl"/>
   <ul>
      <cfoutput query="courseList">
         <li><a href="?coursenum=#coursenum#">#name#</a></li>
      </cfoutput>
   </ul>
<cfelse>
   <cfinvoke
webservice="#REQUEST.prefix#webservices/solutions/coursesWSdone.cfc?wsdl"
method="getOneCourse" returnvariable="onerecord">
      <cfinvokeargument name="coursenum" value="#Val(coursenum)#" />
   </cfinvoke>
   <cfoutput query="oneCourse">
      <p style="font-family:Arial">
         <span style="font-size:18px;font-style:bold">#coursenum# -
#name#</span>
```

```
            <br>
            <span style="font-size:12px">#description#</span>
            <br><br>
            <a href="?">Back to List</a>
        </p>
    </cfoutput>
</cfif>
</body>
</html>
```

Invoking External Web Services

Invoking Web Services is what it's all about! In order to gain access to a Web Service and execute it, you must be aware that it exists.

Locating Web Services

Many Web Services are intended for a small group of users. Perhaps they contain proprietary business information that is intended to be shared with a small group of partners. Others, however, are intended for mass-use. Universal Description, Discovery and Integration (UDDI) is a standard that allows Web Services to be shared.

In fact, a there are many lists of public Web Services available (and certainly many more will be published in the coming years). Here is one:

- http://www.xmethods.com/

Many of these services are free, but most require registration. All requests to these services required some sort of password or key.

Invoking Web Services from .NET, Java and more

As long as Web Services follow the standards in WSDL, SOAP, etc, there is no reason that ColdFusion cannot be used to consume these Web Services. This, in fact, is one of the biggest benefits of Web Services – virtually any language or technology can create a Web Service and virtually any language or technology may consume these Web Services.

Demo: Random Quote Generator

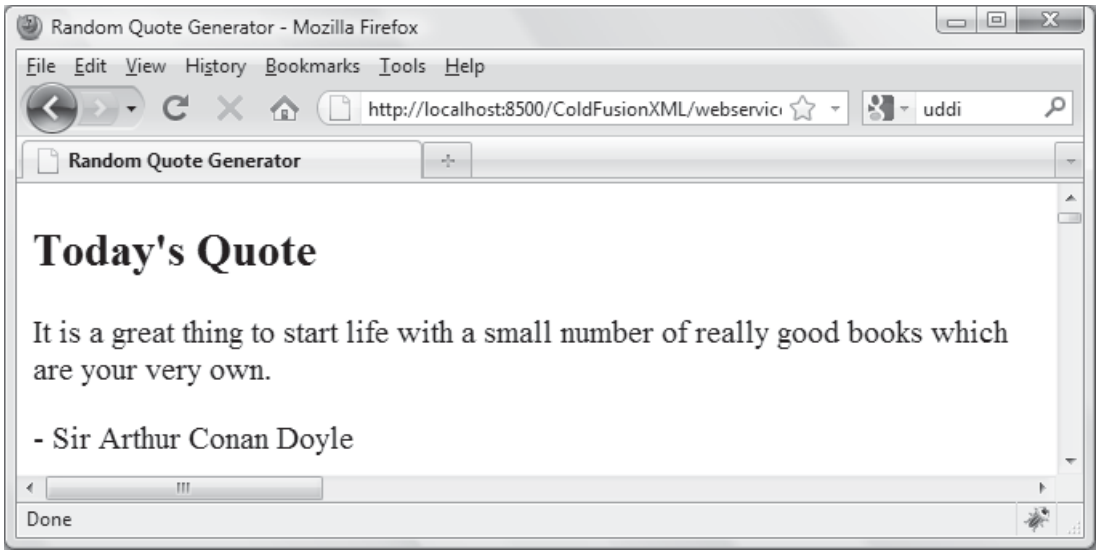
The following demo shows a page which gets a random quote from an External Web Service. It is saved as **webservices/demos/quoteGenerator.cfm**:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Random Quote Generator</title>
</head>
<cfinvoke
webservice="http://www.swanandmokashi.com/HomePage/WebServices/QuoteOfThe
Day.asmx?WSDL"
 method="getQuote"
 returnvariable="aQuote">
</cfinvoke>
```

```
<cfoutput>
    <h2>Today's Quote</h2>
    <p>#aQuote.getQuoteOfTheDay()#</p>
     - #aQuote.getAuthor()#
</cfoutput>

<!---
   Use this <cfdump> tag to show the
   entire object returned.
   <cfdump var="#aQuote#"/>
--->

<body>
</body>
</html>
```



Deeper Examination: <cfinvoke> and <cfinvokeargument>

The quote generator always does the same thing. It returns a random quote. Sometimes it will be useful to pass an argument to a web service. For example, if you pass package id number to FedEx, it could deliver the position of your package.

```
<cfinvoke method="babelFish" returnvariable="translated"
webservice="http://www.xmethods.net/sd/2001/BabelFishService.wsdl">
  <cfinvokeargument name="translationmode" value="en_es" />
  <cfinvokeargument name="sourcedata" value="The rain in Spain falls
mainly on the plain." />
</cfinvoke>
```

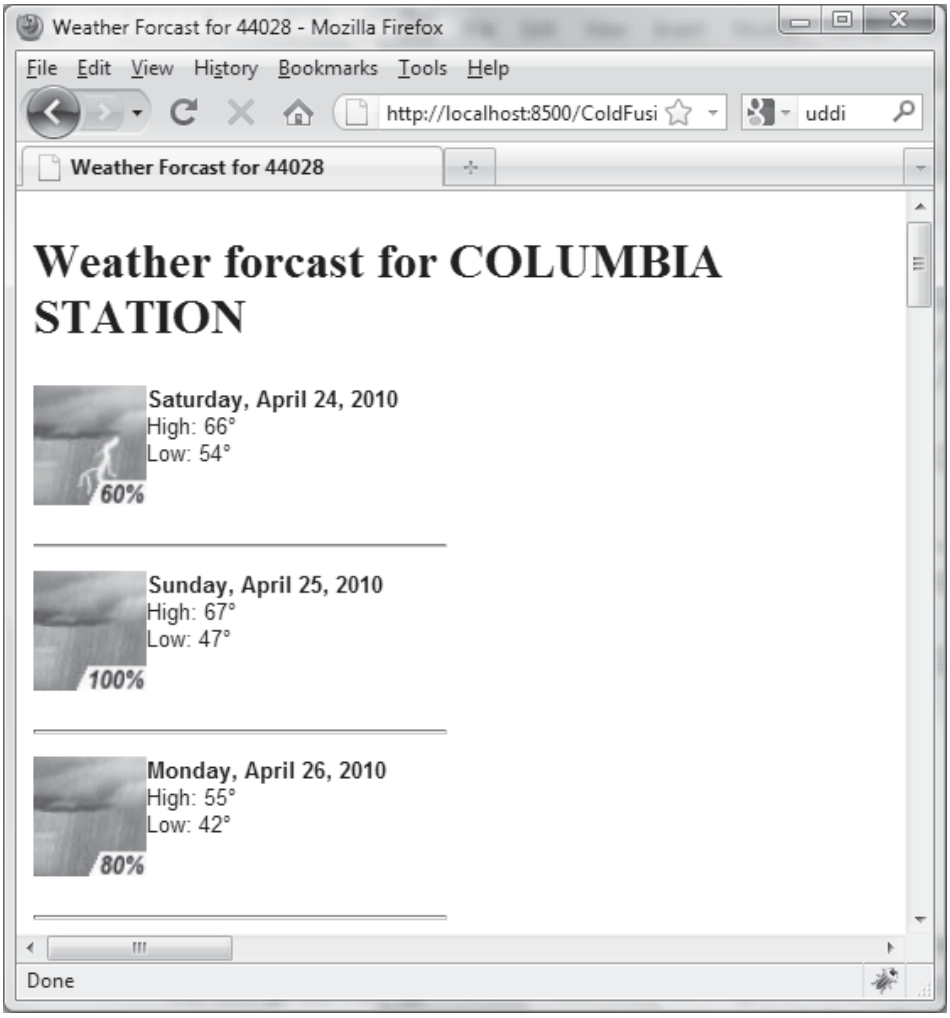
Use **<cfinvokeargument>** to pass arguments to the service.

Exercise 11: Consuming External Web Services

Now it is time for you to consume an web service. Depending on many factors (including your firewall, internet connectivity, corporate policies) you might not be able to reach the web service below. Feel free to find your own web service to try. Remember, many web services require registration. Be sure to read the fine print.

The example below is getting a weather forecast from an external website. You can read more about it on its website: http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=44. Also, the WSDL address is: http://www.webservicex.net/WeatherForecast.asmx?wsdl.

When called in the browser, our page looks like this:



Even the images are customized for each day.

1. Open **webservices\exercises\weatherForecast.cfm**. Follow the three steps in the comments. Add a call to the web service above by adding the following <cfinvoke> tag shown here:

```
<cfinvoke method="getWeatherByZipCode"
  webservice="http://www.webservicex.net/WeatherForecast.asmx?wsdl"
  returnvariable="aWeatherForecasts">
    <cfinvokeargument name="zipCode" value="#URL.zip#"/>
</cfinvoke>
```

2. Drill down into the object that was passed back to receive an array of weather data as shown here:

```
<cfset ArrayofData = aWeatherForecasts.getDetails().getWeatherData()>
```

3. Finally, loop over the array created above to generate the weather forecast using HTML.

```
<cfoutput>
<h2>Weather forecast for #aWeatherForecasts.getPlaceName()#</h2>
<cfloop from="1" to="#ArrayLen(ArrayofData)#" index="x">
    <p></p>
    <p><img src="#ArrayofData[x].getWeatherImage()#"
align="left"/><b>#ArrayofData[x].getDay()#</b><br />
   High: #ArrayofData[x].getMaxTemperatureF()#&deg;<br />
   Low: #ArrayofData[x].getMinTemperatureF()#&deg;</p>
    <br clear="left" />
    <hr width="200" align="left"/>
</cfloop>
</cfoutput>
```

4. Test your page in the browser!

Challenge

- Use <cfdump> to look at the object passed back from the service. Can you drill deep enough to find how to change the forecast to display Celsius.

A Possible Solution to Exercise 11

As contained in **webservices/solutions/weatherForcast.cfm**:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<cfparam name="URL.zip" default="44028"/>
<title>Weather Forecast for <cfoutput>#URL.zip#</cfoutput></title>
<style>
body p { font-family:Arial, Helvetica, sans-serif; font-size:11px}
</style>
</head>
<body>

   <cfinvoke method="getWeatherByZipCode"
     webservice="http://www.webservicex.net/WeatherForecast.asmx?wsdl"
     returnvariable="aWeatherForecasts">
       <cfinvokeargument name="zipCode" value="#URL.zip#"/>
   </cfinvoke>


<!---
Show Entire Object returned
<cfdump var="#aWeatherForecasts#">
--->

<cfset ArrayofData = aWeatherForecasts.getDetails().getWeatherData()>

<cfoutput>
<h2>Weather forecast for #aWeatherForecasts.getPlaceName()#</h2>
<cfloop from="1" to="#ArrayLen(ArrayofData)#" index="x">
    <p></p>
    <p><img src="#ArrayofData[x].getWeatherImage()#"
align="left"/><b>#ArrayofData[x].getDay()#</b><br />
   High: #ArrayofData[x].getMaxTemperatureF()#&deg;<br />
   Low: #ArrayofData[x].getMinTemperatureF()#&deg;</p>
    <br clear="left" />
    <hr width="200" align="left"/>
</cfloop>
</cfoutput>

</body>
</html>
```