Chapter 16.

April 2008

Welcome to the April 2008 edition of IBM Information Server Developer's Notebook. This month we answer the question;

> How do I use invoke an IBM Information Server Web service from within my application development framework?
>
> *Excellent question! In the July/2007 edition of IBM Information Server Developer's Notebook, we detailed how to create and deploy a Web service. Now you're asking how to incorporate (invoke) that Web service from inside your business application.*
>
> In short, we are going discuss all of the relevant terms and technologies above, and provide examples that detail how to deliver this functionality using IBM Information Server.

Software versions

All of these solutions were *developed and tested* on IBM Information Server (IIS) version 8.01, FixPak 1A, using the Microsoft Windows XP/SP2 platform to support IIS client programs, and a RedHat Linux Advanced Server 4 (RHEL 4) FixPak U6 32 bit SMP server (Linux kernel version 2.6.9-67.EL-smp) to support the IIS server components.

IBM Information Server allows for a single, consistent, and accurate view of data across the full width of the corporate enterprise, be it relational or non-relational, staged or live data. As a reminder, the IBM Information Server product contains the following major components;

> WebSphere Business Glossary Anywhere™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Server™, Classic Federation™, Event Publisher™, Replication Server™, Rational Data Architect™, DataMirror Transformation Server™, and others.

Obviously, IBM Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities.

16.1 Terms and core concepts

Web services and IBM Information Server (IIS)

As mentioned above, the July/2007 edition of IBM Information Server Developer's Notebook details how to create and deploy a Web service using IBM Information Server (IIS); at 26 pages, that document also discusses Model-View-Controller, loose coupling, standards for Web services, inter-operability, Service Oriented Architecture (SOA), and more. As an overview from that document, the following is offered;

- Using the IBM Information Server (IIS) DataStage Component, Designer program, we created a DataStage Job that adds two numbers, and returns the result.

  While not the most strategic example, this job carries the basic point; your Web service can do almost anything, and here's how to program (paint) the logic/algorithm for a given Web service.

- Using the IIS Server Console, we then performed the following;
  - Created an Information Services Connection.

    Basically this step identifies the IIS software server that will host our Web service.
  - Created a WISD Project and Application.

    In this context, a Project is a grouping of Applications; an Application equals one IIS DataStage Server Job that will be exposed as a Web service.

    WISD is an acronym for, WebSphere Information Services Director, the component to IIS that provides Web services deployment and management capability.
  - Deployed our WISD Project.
- Using the IIS Web Console, we reverse engineered the URL for our Web service by reviewing that Web service's associated WSDL (Web Services Description Language) document.
- Lastly, using Rational Data Architect (RAD), we opened that tool's Web Services Explorer (WSE), to invoke and explore our deployed Web service. (Any Web services explorer would suffice. We only used RAD because it is part of IIS.)

Application development frameworks

More than just a programming language, or a (programming language) compiler, an *application development framework* extends an application development

environment, into the common structures and design patterns to solve a specified programming challenge. The following is offered;

- If a given application development framework specialized in delivering end user data entry screen forms, that environment might include a single programming language command (or widget, object, whatever) to allow an end user to TAB through specified data entry fields, invoke pop up help, navigate through pick lists of valid data, and so forth.

  For common programming tasks, give the developer an optimized means to deliver this program functionality that is easily maintained, promoted into code re-use, etcetera. For common programming tasks and challenges, make it easy for the developer to be productive.

- Without oversimplifying too much, we will state that there are general purpose programming languages like C or C++, Pascal, Fortran, Basic, etcetera, and then there are purpose built languages like IBM/Informix 4GL, SQL Stored Procedure languages, etcetera.

  Java is a general purpose language. There are packages within (subsets of, or optional and additional components to) Java that form the basis of specific application development frameworks. Struts and Springs are just two examples.

- Web based application development frameworks include;
  - Java/J2EE, including possibly Faces/JSP, Ajax, Struts, Springs, and others.
  - Microsoft's ASP.NET platform, including C# and VB.Net.
  - Ruby on Rails, PhP, and Python.
  - And dozens more.

Note: Part of the value proposition of Web services is their inter-operability; Web services are a defacto standard, and are available for use with every modern and progressive application development framework. While Web services definition, delivery, consumption, etcetera, are standard, each programming language may have its own specific command or object to invoke a Web service. Just as a given language construct to, for example, count from 1 to 10 is different in C, or Java, or anything else, so too may be the specific command to invoke a Web service.

In effect then, this month's question of how to invoke a Web service is platform specific. The platform we choose to answer this question in is Ajax, using a simple desktop Web browser, minimal HTML and related program code. The example we provide serves to answer most questions. If you need to implement, for example, a PhP specific solution, there remains a smallish amount of PhP specific code you will need to research.

16.2 What is Ajax?

Ajax is not an acronym, but it could be. Ajax is short for Asynchronous JavaScript and XML. Ajax is a core technology behind Google Suggest, Google Maps, Amazon's search engine A9.com, Yahoo! News, and more. Ajax can be used to support drag and drop within a Web page, auto-complete of text entry fields, specialized menus, floating widgets and frames, and more.

> *Imagine if you dragged the currently displayed book on your Amazon.com search page onto a graphic for your shopping cart, picking the given item, and all of this occurred and was updated without having to re-display/re-render the page? At the same time, you could modify the shipping method, recalculating the total order cost, again without re-serving the page- Ajax does stuff like that.*

In our solution, we are going to use Ajax to directly invoke a Web service; reading from two text entry fields inside the Web form, and returning a result calculated from those source values. (The add two numbers Web service from the July/2007 edition of IBM Information Servers Developer's Notebook.) In the real world, the Web service would most likely be invoked on the server side of any application development framework you were using, and values sent and received would be placed automatically into any Web browser resident visual controls, (text entry fields and the like).

One advantage towards using Ajax in the manner offered below, is that this solution will work inside any Web browser without requiring a framework; this solution works without Java, PhP, C#, etcetera. All this solution needs is a Web browser.

Figure 16-1 displays a simple HTML Web page we create as a solution in this document. While the presentation is simple HTML, an IBM Information Server hosted Web service retrieves the sum of the two text entry field column values. The "onClick" event of the "Get Result" command button invokes an IIS Web Service using a JavaScript object entitled, XMLHttpRequest. XMLHttpRequest returns the result of the Web service asynchronously, and, it does so without having to re-display or re-render anything in the browser.

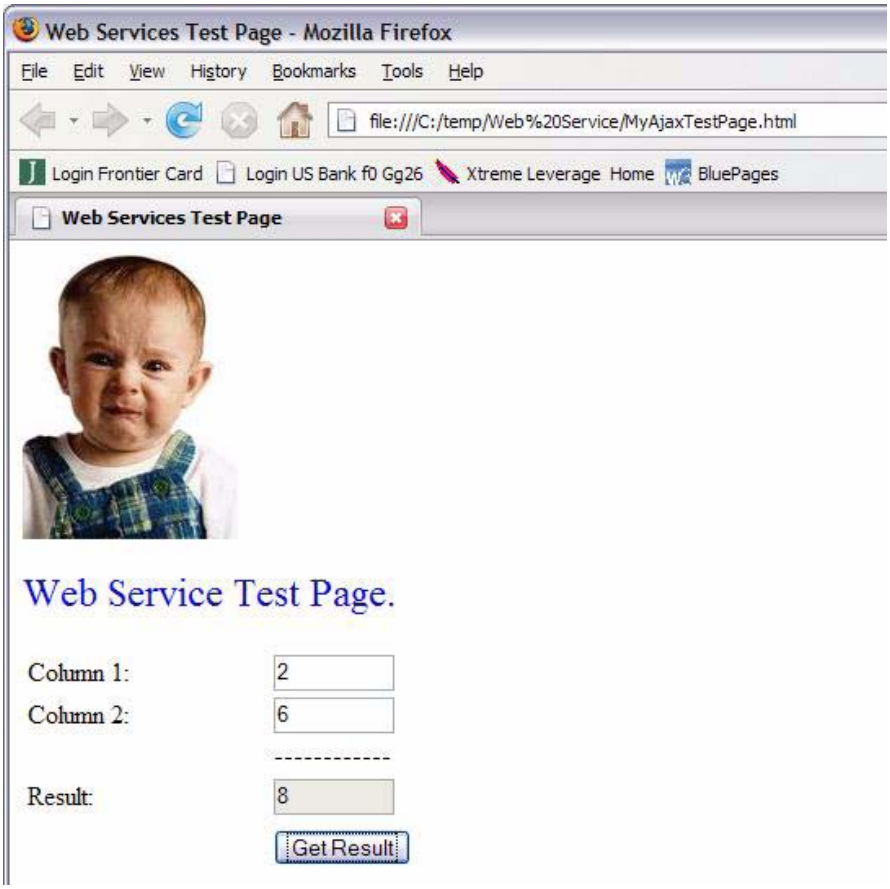

Figure 16-1 Finished example, what we are about to create.

More on Ajax

Model-View-Controller (MVC) is a *design pattern*, a concept as to how one should organize and arrange computer program source code. MVC calls to separate the end user presentation layer (the View), from the data persistence layer (the [data] Model), and ties these together with the Controller, program code in the middle that associates end user initiated program events with database calls. One advantage of MVC is that all database code is contained within a specific set of files that form the entire application. A change to just the database should only affect this subset of the program code, easing program maintenance and quality assurance burdens, and other concerns. Similarly, a change to the user interface affects only that subset of the code.

Whereas a design pattern is a concept, or a logical entity, an *application development framework* is a physical entity. A framework can include (libraried) program source code, a hierarchy of objects to deliver given functionality, perhaps tooling, etcetera. Faces/JSP, for example, is referred to as a *server side framework*, and Faces/JSP supports the MVC design pattern. As each Web browser based event sends a request to the application server, Faces/JSP creates a DOM (*document object model*, in this case, a hierarchical representation of the HTML page data), which in turns invokes the correct server side function on the application server associated with the given event. This is all done via the framework of Faces/JSP and the application server runtime.

Ajax is also a framework that supports MVC. However, Ajax is a *client side framework* (not necessarily a problem) with its own DOM (problem). The Ajax DOM and Faces/JSP DOM are dis-similar.

Ajax uses the HTTP(S)/SOAP communication protocol, and can be used to send XML or HTML formatted messages. In our solution, we are going to have Ajax invoke an IBM Information Server (IIS), DataStage component authored Web service. Being able to bridge these two (frameworks, Ajax and DataStage), is a further testament to MVC and the power of Web services; In this case, the View will be an HTML form with Ajax, and the Controller-Model will be an IIS DataStage authored Web service. (The communication protocol is HTTP(S)/SOAP, and all messages will be XML formatted.)

Note: Faces/JSP is an extensible framework. While it would require Java/J2EE programming knowledge, one could create a new custom Faces/JSP visual component that would use Ajax. For example, earlier it was mentioned that Ajax can perform auto-complete of text entry fields. To deliver this capability with Faces/JSP, one would most likely program an extension to an existing Faces/JSP text entry field, and supply the appropriate Ajax (Javascript) code to manage the client side operations. Faces/JSP would manage the server side, finding the text value to auto-complete, and managing the communication to the client.

From an everyday programmer stand point, this new (custom) Faces/JSP visual control would look like any other Faces/JSP visual control. In fact, numerous standard Faces/JSP visual controls contain client side Javascript program code.

More on Web Services, What and Why?

Continuing with the concept of Model-View-Controller, one would normally expect that a given end user program is still compiled into a single binary file of some sort, like it has been for 30 years, and that this binary file is executed via some means thus supplying our program functionality.

With Web services, the entire end user program is not compiled into one executable. Parts of the program are invoked via remote procedure call (RPC), also known as remote method invocation (RMI). As the name implies, this Web service could be remote, hosted on an entirely different type of computer. Web services differ from shared libraries or dynamically linked libraries (on Unix/Linux, ".so" files, and on Windows ".dll" files), in that they are never linked. Linking would require local copies of these methods, and compatible binary editions (same language and compiler, if you will).

Web services can run over the HTTP(S) protocol, more specifically HTTP(S)/SOAP, which further defines the remote messaging protocol. A WSDL file (Web Services Description Language), which is ASCII text but formatted in XML, is used to describe the Web service being published or consumed. A Web service can be invoked via an absolute address, or it can be located via a call to a UDDI (Universal Description, Discovery, and Integration Protocol) registry. In short, a call can be made to a UDDI registry describing the type of service you want to receive, without knowing much about the Web service provider, or specific implementation details. The last of the Web services related acronyms is WSIL (Web Services Inspection Language). WSIL is a different and lower cost means to deliver UDDI related functionality.

Web services have many benefits. Following the concept of MVC, one could upgrade their user interface (View) portion of their application without having to re-deploy their Controller or (data) Model portions. And, as is the case in the continuing example in this document, the Web service requestor and Web service provider could be using different (or even incompatible) technologies.

16.3 Example, Ajax invoking an IIS Web service

Example 16-1 offers the complete HTML and Ajax source code example needed to invoke a Web service from inside a Web browser. Following Example 16-1, a code review is offered.

Example 16-1 Complete HTML and Ajax source code example to invoke Web service.

```
<!-- tpl:insert page="/theme/A_gray.htpl" --><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="Content-Style-Type" content="text/css">
<!-- tpl:put name="headarea" -->
<title>Web Services Test Page</title>
```



```
<SCRIPT type="text/javascript">

  function func_1(thisObj, thisEvent) {



  netscape.security.PrivilegeManager.
    enablePrivilege("UniversalBrowserRead");

  var myCol1 = document.getElementById("col1");
  var myCol2 = document.getElementById("col2");
  var myCol3 = document.getElementById("col3");
    //
  var myServerURL = 'http://192.168.1.10:9080/' +
    'wisd/AddTwoNumbers_App/AddTwoNumbers_Svc';

  var mySOAPMsg =
    '<?xml version="1.0" encoding="UTF-8" ?>' +
    '<soapenv:Envelope' +
    ' xmlns:q0="http://AddTwoNumbers_Svc.AddTwoNumbers_App.' +
    'isd.ibm.com/soapoverhttp/"' +
    ' xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"' +
    ' xmlns:xsd="http://www.w3.org/2001/XMLSchema"' +
    ' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' +
    '<soapenv:Body>' +
    '   <q0:addTwoNumbers_Method>' +
    '      <col1>' +
            myCol1.value +
    '      </col1>' +
    '      <col2>' +
            myCol2.value +
    '      </col2>' +
    '   </q0:addTwoNumbers_Method>' +
    '</soapenv:Body>' +
    '</soapenv:Envelope>' ;
```



```
var myRequest = new XMLHttpRequest();   // Specific to Firefox, need to
                                        // change for IE.

myRequest.open("POST", myServerURL, true);
  //
myRequest.setRequestHeader("Content-Type", "text/xml");
myRequest.setRequestHeader("SOAPAction", myServerURL);
  //
myRequest.send(mySOAPMsg);



myRequest.onreadystatechange=function(){
  if (myRequest.readyState == 4) {
    if (myRequest.status == 200) {



      //  var myResponseAllTold  =
      //     myRequest.responseXML.getElementsByTagName(
      //     "addTwoNumbers_MethodReturn");
      //  var myResponseJustData =
      //     myResponseAllTold[0].firstChild.data;
      //        //
      //  myCol3.value = (myResponseJustData);



      var myResponseAsText = myRequest.responseText;
        //
      var prePart  = myResponseAsText.substr(
        myResponseAsText.indexOf("MethodReturn>") + 13);
      var postPart = prePart.substr(0, prePart.indexOf("<"));
        //
      myCol3.value = (postPart);
    } else {
      alert('ERROR: ' + myRequest.responseText );
    }
  }
```



```
  }



}
</SCRIPT><!-- /tpl:put -->


</head>
<body>
<table width="760" cellspacing="0" cellpadding="0" border="0">
  <tbody>
    <tr>
      <td valign="top">
      <table class="header" cellspacing="0" cellpadding="0"
          border="0" width="100%">
        <tbody>
          <tr>
            <td width="150">
              <img border="0"
                alt="Me in real life."
                src="CryingGirl.jpg">
            </td>
            <td>
            </td>
          </tr>
        </tbody>
      </table>
      </td>
    </tr>
    <tr>
      <td valign="top" class="nav_head" height="20">
      </td>
    </tr>
    <tr class="content-area">
      <td valign="top" height="350">
        <!-- tpl:put name="bodyarea" -->
        <FONT size="+2" color="blue">Web Service Test Page.</FONT>
```



```
<BR>
<BR>
<TABLE width="60%" border="0">
  <TBODY>
    <TR>
      <TD width="148">Column 1:
      </TD>
      <TD width="302">
        <INPUT type="text" name="col1"
          size="8" maxlength="8" value="0"
          id="col1">
      </TD>
    </TR>
    <TR>
      <TD width="148">Column 2:
      </TD>
      <TD width="302">
        <INPUT type="text" name="col2"
          size="8" maxlength="8" value="0"
          id="col2">
      </TD>
    </TR>
    <TR>
      <TD width="148">
      </TD>
      <TD width="302">----------
      </TD>
    </TR>
    <TR>
      <TD width="148">Result:
      </TD>
      <TD width="302">
        <INPUT type="text" name="col3"
          size="8" maxlength="8" value="0"
          readonly id="col3">
      </TD>
```



```
            </TR>
            <TR>
              <TD width="148">
              </TD>
              <TD width="302">
              </TD>
            </TR>
            <TR>
              <TD width="148">
              </TD>
              <TD width="302">
                <INPUT type="button" name="actionButton"
                  value="Get Result"
                  onclick="return func_1(this, event);">
              </TD>
            </TR>
            <TR>
              <TD width="148">
              </TD>
              <TD width="302">
              </TD>
            </TR>
          </TBODY>
        </TABLE>
        <BR>
        <BR>
        <BR>
        <!-- /tpl:put --></td>
      </tr>
      <tr>
        <td valign="top" class="footer" height="20">
        </td>
      </tr>
    </tbody>
  </table>
  </body>
```



```
</html>

<!-- /tpl:insert -->
```

While the example in Example 16-1 may seem long, there are really only two areas of functionality that we need to understand. The rest is just HTML formatting and standard HTML document setup.

Using pure HTML, three input text fields, and one push button are defined. Example as shown in Example 16-2;

Example 16-2 Subset of Example 16-1, just the input text fields and button.

```
<INPUT type="text" name="col1" size="8" maxlength="8" value="0" id="col1">

{lines deleted}

<INPUT type="text" name="col2" size="8" maxlength="8" value="0" id="col2">

{lines deleted}

<INPUT type="text" name="col3" size="8" maxlength="8" value="0" readonly id="col3">

{lines deleted}

<INPUT type="button" name="actionButton" value="Get Result" onclick="return func_1(this, event);">
```

A code review of Example 16-2 is offered below;

- Example 16-2 is pure HTML, and could be painted using any HTML editing tool.
- The first three input text fields are given the id's, col1, col2 and col3. col1 and col2 are used as input values to our Web service; col3 will hold the result.
- col3 is defined as read only, because it will only hold a result and should not be edited by the end user.
- actionButton invokes a client side JavaScript function when clicked.

Example 16-3 completes the source code review, offering the JavaScript function invoked by the button above.

Example 16-3 JavaScript function to invoke our Web service.

```
function func_1(thisObj, thisEvent) {



  netscape.security.PrivilegeManager.
    enablePrivilege("UniversalBrowserRead");


  var myCol1 = document.getElementById("col1");
  var myCol2 = document.getElementById("col2");
  var myCol3 = document.getElementById("col3");
    //
  var myServerURL = 'http://192.168.1.10:9080/' +
    'wisd/AddTwoNumbers_App/AddTwoNumbers_Svc';


  var mySOAPMsg =
    '<?xml version="1.0" encoding="UTF-8" ?>' +
    '<soapenv:Envelope' +
    ' xmlns:q0="http://AddTwoNumbers_Svc.AddTwoNumbers_App.' +
    'isd.ibm.com/soapoverhttp/"' +
    ' xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"' +
    ' xmlns:xsd="http://www.w3.org/2001/XMLSchema"' +
    ' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' +
    '<soapenv:Body>' +
    '   <q0:addTwoNumbers_Method>' +
    '      <col1>' +
             myCol1.value +
    '      </col1>' +
    '      <col2>' +
             myCol2.value +
    '      </col2>' +
    '   </q0:addTwoNumbers_Method>' +
    '</soapenv:Body>' +
    '</soapenv:Envelope>' ;
```



```
var myRequest = new XMLHttpRequest();   // Specific to Firefox, need to
                                        // change for IE.

myRequest.open("POST", myServerURL, true);
  //
myRequest.setRequestHeader("Content-Type", "text/xml");
myRequest.setRequestHeader("SOAPAction", myServerURL);
  //
myRequest.send(mySOAPMsg);



myRequest.onreadystatechange=function(){
  if (myRequest.readyState == 4) {
    if (myRequest.status == 200) {



      //  var myResponseAllTold  =
      //     myRequest.responseXML.getElementsByTagName(
      //     "addTwoNumbers_MethodReturn");
      //  var myResponseJustData =
      //     myResponseAllTold[0].firstChild.data;
      //        //
      //  myCol3.value = (myResponseJustData);



      var myResponseAsText = myRequest.responseText;
        //
      var prePart  = myResponseAsText.substr(
        myResponseAsText.indexOf("MethodReturn>") + 13);
      var postPart = prePart.substr(0, prePart.indexOf("<"));
        //
      myCol3.value = (postPart);
    } else {
      alert('ERROR: ' + myRequest.responseText );
    }
  }
```



```
  }

}
```

A code review of Example 16-3 is offered below;

- The above (and full) example was tested to work in FireFox, version 2.0.0.15, on a Microsoft Windows client, attaching to a Linux server. Some of the JavaScript program code is platform specific; meaning, it wont work in Microsoft Internet Explorer without modification. If you Google the specific keyword as documented, and Internet Explorer, the answer should be easy to locate; these topics are all widely known regarding Web browser compatibilities.
- The line containing the keyword "netscape" calls to enable a given Web browser security method;
  - The method is required in newer versions of Web browsers.

    This method is also required when running this example in Client/Server. When running locally, this method invocation is not required.
  - The first time you run this example, the Web browser will throw a warning message; if you Click Okay, you'll never see this warning again.
- The getElementById give us a pointer to each of the named controls, allowing us to get and set each control's value property.
- Where we set the server URL, that line is hard coded and will need to be set per your installation.

  In a more complete example, we could make this value dynamic via a variety of methods. The AddTwoNumbers_App/AddTwoNumbers_Svc was specific to our deployed IIS DataStage service. The remainder of that string is default.
- The var_SOAPMsg is pretty standard.

  We got this value from the Rational Data Architect Web Services Explorer client program as detailed in the July/2007 edition of the IBM Information Server Developer's Notebook document.
- The XMLHttpRequest line is specific to FireFox and needs to change for Internet Explorer.
- The myRequest.open method is the line that causes the netscape warning detailed above.

- The myRequest.send method invocation is done asynchronously, and is a non-blocking call. This is determined by the 'true' argument to myRequest.open.
- The JavaScript inline function, onreadystatechange, is the function that is called asynchronously, in response to a successful and complete XMLHttpRequest response.

  The remainder of this JavaScript source code is encased in this inline JavaScript function, and is detailed below;

  - myRequest.readyState and myRequest.status are true when the XMLHttpRequest response is complete and successful.
  - The lines prefaced with // are commented out, and are the preferred syntax we would wish to perform.

    This source code is commented out because a current Web browser security standard dis-allows execution of the getElementsByTagName inside an XMLHttpRequest asynchronous request.

    In effect, you can't dynamically parse an XML document. You can parse that very same response as text, but not as an XML document.

    This commented code works in older Web browsers, not the current release of FireFox though.
  - The prePart and postPart lines parse the XML response as text and using standard JavaScript substring processing; effective, not as elegant as XML node processing.
  - And then we set the column value for col3, our results text entry field.

Using your favorite line editor, or HTML development environment, create a single, sample HTML file similar to that as displayed in Example 16-1. Run this file in a Web browser, and retrieve the results of an IBM Information Server provided Web service. (Or any server provided Web service, it all works.)

Enhance your example to be more dynamic, offering less hard coding.

16.4 Summation

In this document, we reviewed or created:

We detailed invoking a Web service using Ajax and a standard HTML form. The Web service we tested was provided by IBM Information Server, from an example first offered in July/2007, in this very same publication.

While we used Ajax because of its platform (application development framework) independence, every modern development framework we can think of has this capability in some form or another.

Persons who help this month.

Jonathan Sayles.

Additional resources:

None.

Legal statements:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating trademarks that were owned by IBM at the time this information was published. A complete and current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

Other company, product or service names may be trademarks or service marks of others.

Special attributions:

The listed trademarks of the following companies require marking and attribution:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Microsoft trademark guidelines

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel trademark information

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Other company, product, or service names may be trademarks or service marks of others.

