June 08 | Volume 1 |Issue 12 | Free

# Java Jazz up

A BETTER WAY TO LEARN PROGRAMMING

# Web Services

*Introduction to web services*
*Google Web Services*
*Amazon Web Services*
*Web Services and Ajax*
*SOA and Web Services*
*Implementing WS-Security*

# India's Cheapest web Service Provider

RoseIndia

## OUR SERVICES

- ERP Soluiotns
- Software Solutions
- Web Development
- Web Designing
- Web Redesigning
- Domain Registration
- Web Promotion
- SEO
- Article Writing
- Blog Writing
- News Writing
- SEO Copywriting
- Technical Documentation
- E-Commerce Solutions
- CRM
- Outsourcing

Extend your reach with our solutions...

http://www.roseindia.net/services

Java Jazz up
A BETTER WAY TO LEARN PROGRAMMING

June 2008 Volume I Issue XII

**"Optimism with determination lets you hit the goal harder"**

**Published by**

**RoseIndia**





## Editorial

Dear Readers,

We are back here with the June 2008 special issue of Java Jazz-up. This current June edition is designed as 'Web Services' special issue. From some previous issues we decided to release our magazine as special issue on a single technology. Our last issue was designed as 'Web Services' special issue. The issue really proved very popular and valuable among our readers. The current issue extends 'Web Services' initialized from the last issue. We hope the issue will be more popular and valuable for all readers.

Web Service is one of the most popular and useful techniques in the landscape of software development. Web Services allows different applications to talk to each other and share data and services among them.

Other applications can also use the services of the web services. For example VB or .NET application can talk to java web services and vice versa. So, Web services are used to make the application platform and technology independent. In the issue, our readers will learn about web services providers, RESTful web services, SOA and web services, open source web services tools, web services and ajax, WS-Security etc.

In addition as per the growing popularity of Web Services we have included four relevant tutorials on Web Services published in IBM developerWorks with consent. We are thankful to the team of IBM for granting permission regarding the same.

To make it interesting for readers we have categorized each section with different colors and images that would certainly lure readers while reading technological stuffs. We are providing it in PDF format that you can view and even download it as a whole or a part. This PDF version provides you a different experience.

Please send us your valuable feedback about this issue and participate in the reader's forum with your problems and issues concerned with the topics you want us to include in our next issues.

**Editor**
Deepak Kumar
Java Jazz up

# Content


**05** **Quick introduction to web services |** Web services are becoming more and more popular. Web Services allows you to expose the functionality of your existing code over the network.

**07** **Amazon Web Services (AWS)|** Everyone knows that Amazon is one of the biggest online shopping store selling wide variety of products like books, magazines, DVDs, videos, electronics, computers, software, apparel & accessories, shoes etc.

**08** **Google web Service (GWS) |**Most users spend their time with Google while using Internet. Whenever we have to search something we select Google for it.

**09** **RESTful Web Services |** RESTful Web services are very popular currently. REST (Representational State Transfer) is an alternative to SOAP, XML-RPC etc., that provides simple, platform-neutral data exchange using the HTTP transport. It can be integrated with HTTP more efficiently than SOAP based services.

**10** **SUA and Web Services |**Service Oriented Architecture (SOA) is a new architecture for the development of loosely coupled distributed applications. In fact service-oriented architecture is collection of many services in the network.

**12** **Open Source web services tool in java |** The intention for this project is a very simple API to call different kinds of services (provider/technology). Crispy's aims is to provide a single point of entry for remote invocation for a wide number of transports: eg. RMI, EJB, JAX-RPC or XML-RPC.

**16** **Automate data entry with Web services and Ajax|** The United States Postal Service (USPS) has made several Web services available (see the USPS Web tools sidebar). One of those Web services accepts a ZIP code and returns the corresponding city and state.

**27** **Implementing WS-Security|** This article describes how the emerging WS-Security standard was used to secure a Web service that was developed and deployed in the fall of 2002. The article will discuss the security-related requirements of the Web service and how they were met using a combination of HTTPS/SSL, digital certificates, and digital signature technologies.

**36** **Cross-platform programming with Java technology and the IBM Web Services Toolkit for Mobile Devices|** When you're writing a Java application for the server or desktop, you can be reasonably sure that the Java platform will fulfill its "Write Once, Run Anywhere" promise. But when you're dealing with code that will run under the J2ME Mobile Information Device Profile (MIDP), things get a little trickier.

**41** **Understanding quality of service for Web services Improving the performance of your Web services|** With the widespread proliferation of Web services, quality of service (QoS) will become a significant factor in distinguishing the success of service providers. QoS determines the service usability and utility, both of which influence the popularity of the service.

**50** **Advertise with Us |** We are the top most providers of technology stuffs to the java community. Our technology portal network is providing standard tutorials, articles, news and reviews on the Java technologies to the industrial technocrats.

**51** **Valued JavaJazzup Readers Community |** We invite you to post Java-technology oriented stuff. It would be our pleasure to give space to your posts in JavaJazzup.

# Quick introduction to web services

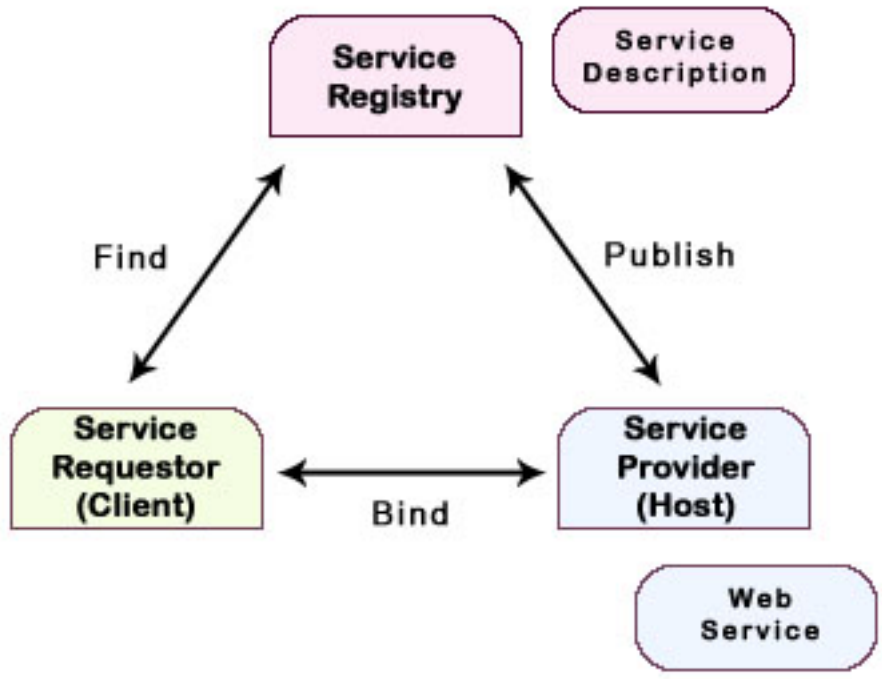
**1. Introduction:**

Web services are becoming more and more popular. Web Services allows you to expose the functionality of your existing code over the network. Once it is exposed on the network, other application can use the functionality of your program.

Web Services allows different applications to talk to each other and share data and services among them. Other applications can also use the services of the web services. For example VB or .NET application can talk to java web services and vice versa. So, Web services are used to make the application platform and technology independent.

These days due to complexness the business, organizations are using different technologies like EAI, EDI, B2B, Portals etc. for distributing computing. Web Services supports all these technologies, thus helping the business to use existing investments in other technologies.

One of the major benefits is Web services' ease of integration. You will easily integrate your application with other pieces of software application i.e. with external data sources. You can run it on all kinds of machines from desktop to mainframe and from within your enterprise to external sites i.e. Web services link applications, services, and devices together.




In the above figure "Service Requester" acts as client of the web service and the "Service Provider" as host of the web service. Client makes a request for the web service to the provider and provider sends response accordingly.

## Quick introduction to web services

Service registry acts as a database of web services containing all the information needed to develop a web client, the definition and uri for web services. You can publish your services to the registry. Now any client can query the registry and select the service from there.

# Amazon Web Services (AWS)

**2. Amazon Web Services (AWS):**

Everyone knows that Amazon is one of the biggest online shopping store selling wide variety of products like books, magazines, DVDs, videos, electronics, computers, software, apparel & accessories, shoes etc. Amazon provides friendliest shopping experience online to their customers and excellent customer service.

Amazon is known for its creative and innovative ideas that simplify and enhance the customer experience. Amazon web services (AWS) are one of those new ideas. AWS provides a set of APIs to query the complete Amazon database with the help of SOAP-based calls.

Now Amazon Web Services (AWS) released a wide variety of web services that provides an easy and inexpensive access to Amazon's robust infrastructure. Using AWS, developers can build reliable, scalable, and cost-effective web applications.

**List of AWS:**

- Alexa Top Sites
- Amazon DevPay
- Alexa Web Search
- Alexa Site Thumbnail
- Alexa Web Information Service
- Amazon SimpleDB
- Amazon Associates Web Service (Formerly known as Amazon E-Commerce Service (ECS))
- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon Simple Storage Service (Amazon S3)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon Mechanical Turk (Amazon MTurk)
- Amazon Historical Pricing
- Amazon Flexible Payments Service (Amazon FPS)
- Amazon Fulfillment Web Service (Amazon FWS)

**Url: http://aws.amazon.com/**

# Google web Service (GWS)

Most users spend their time with Google while using Internet. Whenever we have to search something we select Google for it. Google provides its search functionality as its web service. In your application, you can add the power of Google search engine with the help of Google Web Service. Google web service allows any developer to access search, cache and spell check services.

**Url: http://code.google.com/**

# RESTful Web Services

RESTful Web services are very popular currently. REST (Representational State Transfer) is an alternative to SOAP, XML-RPC etc., that provides simple, platform-neutral data exchange using the HTTP transport. It can be integrated with HTTP more efficiently than SOAP based services.

RESTFul style can be good when web services are fully stateless, caching can be good for performance, service consumer and producer have an understanding of the context, and bandwidth is important and limited. Main advantages of REST web services are lightweight, human readable results and easy to build.

RESTful systems map very closely to CRUD based-data operations. Using HTTP methods like PUT, POST, GET, DELETE and a URI, the server takes action on the request. GET is used to query on a resource. POST is used to create a new resource. PUT is used to update the resource. DELETE is used to remove a resource.

**Read more at http://java.sun.com/developer/technicalArticles/WebServices/restful/**

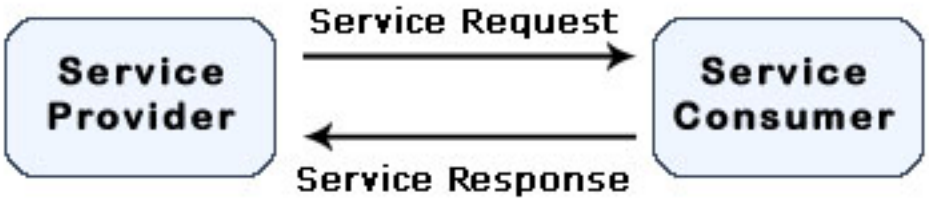
# SUA and Web Services

Service Oriented Architecture (SOA) is a new architecture for the development of loosely coupled distributed applications. In fact service-oriented architecture is collection of many services in the network. These services communicate with each other and the communications involves data exchange & even service coordination. Earlier SOA was based on the DCOM or Object Request Brokers (ORBs). Nowadays SOA is based on the Web Services. Broadly SOA can be classified into two terms: Services and Connections.

**Services:**
A service is a function or some processing logic or business processing that is well defined, self-contained, and does not depend on the context or state of other services. Example of Services is Loan Processing Services, which can be self-contained unit for process the Loan Applications. Other example may be Weather Services, which can be used to get the weather information. Any application on the network can use the service of the Weather Service to get the weather information.

**Connections:**
Connections means the link connecting these self-contained distributed services with each other, it enable client to Services communications. In case of Web services SOAP over HTTP is used to communicate the between services.

The following figure is a typical example of the service-oriented architecture. It shows how a service consumer sends a service request to a service provider. After accepting the request, service provider sends a message to the service consumer. In this case a service provider can also be a service consumer.




**Different Technologies Used:**

SOA is much different from point-to-point architectures. SOA comprise loosely coupled, highly interoperable application services. These services can be developed in different development technologies (such as Java, .NET, C++, PERL, PHP), the software components become very reusable i.e. the same C# (C Sharp) service may be used by a Java application and / or any other programming language. WSDL defines an standard, which encapsulates / hides the vendor / language specific implementation from the calling client / service.

**Why SOA?**

SOA architecture enables seamless Enterprise Information Integration. Here are some of the Benefits of the Service Oriented Architecture:

1. Due to its platform independence, it allows companies to use the software and hardware of their choice

2. There is no threat of vendor lock-in
3. SOA enables incremental development, deployment, and maintenance.
4. Companies can use the existing software (investments) and use SOA to build applications without replacing existing applications
5. The training costs are low, so the available labor pool can be used for running the applications

# Open Source web services tool in java

1. **Apache Axis :**
**Url**: http://ws.apache.org/axis/
**About Axis**: Apache Axis is an implementation of the Simple Object Access Protocol (SOAP).

2. **WSIF** :
**Url** : http://ws.apache.org/wsif/
**About WSIF**:
The Web Services Invocation Framework (WSIF) is a simple Java API for invoking Web services, no matter how or where the services are provided.

3. **Metro** :
**Url** : https://metro.dev.java.net
**About Metro**:
Metro is a high-performance, extensible, easy-to-use web service stack. It is a one-stop shop for all your web service needs, from the simplest hello world web service to reliable, secured, and transacted web service that involves .NET services.

4. **Crispy** :
**Url** : http://crispy.sourceforge.net/
**About Crispy**:
CRISPY = Communication per Remote Invocation for different kinds of Services via ProxYs.

The intention for this project is a very simple API to call different kinds of services (provider/ technology). Crispy's aims is to provide a single point of entry for remote invocation for a wide number of transports: eg. RMI, EJB, JAX-RPC or XML-RPC. It works by using properties to configure a service manager, which is then used to invoke the remote API. Crispy is a simple Java codebase with an API that sits between your client code and the services your code must access. It provides a layer of abstraction to decouple client code from access to a service, as well as its location and underlying implementation. The special on this idea is, that these calls are simple Java object calls (remote or local calls are transparent).

5. **Apache Scout**:
**Url** : http://ws.apache.org/scout
**About Apache Scout**:
Apache Scout is an implementation of the JSR 93 - JavaTM API for XML Registries 1.0 (JAXR).

6. **Apache Axis2** :
**Url** : http://ws.apache.org/axis2/
**About Apache Axis2**:
Axis2 is Apache's Web services framework with two implementations Apache Axis2/Java and Apache Axis2/C. As discussed earlier Web services are very good means of inter application communication. This communication is possible with the help of XML. SOAP protocol defines the schema for a message to be sent when using web services. SOAP engine is required to create and interpret SOAP messages. AXIS2 is such a SOAP engine.

7. **Xfire** :
**Url** : http://xfire.codehaus.org/
**About Xfire**:
Codehaus XFire is a next-generation java SOAP framework. Codehaus XFire makes service oriented development approachable through its easy to use API and support for standards. It is also highly performant since it is built on a low memory StAX based model.

8. **Caucho Hessian** :
**Url** : http://www.caucho.com/hessian/
**About Caucho Hessian**:
The Hessian binary web service protocol makes web services usable without requiring a large framework, and without learning yet another alphabet soup of protocols. Because it is a binary protocol, it is well-suited to sending binary data without any need to extend the protocol with attachments.

9. **SOAP UDDI** :
**Url** : http://soapuddi.sourceforge.net/
**About SOAP UDDI**:
Implementation of UDDI 2.0 compliant registry. This is a reference implementation of UDDI specification. With this UDDI Registry, Web services developers can publish and test their applications in a secure, private environment for their own applications.

10. **Ivory** :
**Url** : http://ivory.codehaus.org/
**About Ivory**:
Ivory provides easy integration between your exiting java classes, Avalon services, and Axis. It allows easy deployment of soap services with none of the WSDD configuration that Axis normally mandates.

11. **XINS** :
**Url** : http://xins.sourceforge.net
**About XINS**:
XINS is an open-source Web Services framework supporting HTTP protocols such as REST, SOAP, XML-RPC, JSON, JSON-RPC and more.
From the specifications written in simple XML, XINS generates the Client API (.jar), the Java server code template (.war), the WSDL and the documentation of the specification in HTML (with the test forms) or in OpenDocument format.

12. **Caucho Burlap** :
**Url** : http://www.caucho.com/burlap/
**About Caucho Burlap**:
Burlap is a simple XML-based protocol for connecting web services. The com.caucho.burlap.client and com.caucho.burlap.server packages do not require any other Resin classes, so can be used in smaller clients, like applets.
Because Burlap is a small protocol, J2ME devices like cell-phones can use it to connect to Resin servers. Because it's powerful, it can be used for EJB services.

**News:**

**1. CompuSystems Offers Web Services Data Exchange to Facilitate System Integration:**

CompuSystems, Inc. has announced a service to improve the speed and flexibility of updating event and show organizer databases.
According to Chris Williams, CompuSystems' Sr. VP of Sales: 'Web services and integration come up in nearly every client and prospect presentation. I think we'll integrate our systems with the systems of every association client we have within the next five years. It's one of the most significant topics in registration at the moment, and I don't see that changing until it begins to be utilized more widely.'

Rob Sarkis, CompuSystems' CIO and VP of Information Technology said, 'Association clients today are more sophisticated in the way they collect and use data. Integrating systems allows faster membership file updating and a more efficient registration process; web services enables that in real time.'

Paul McCaffray, CompuSystems' Executive VP and COO said, 'Web services has provided the ability to leverage the collective data, eliminate duplicate data entry, and deliver a truly integrated registration service.'

**2. VisualCV Inc. creating an online multimedia resume portal service:**

VisualCV Inc. is creating an online multimedia resume portal service utilizing cloud computing from Amazon.com.

According to Scott Herman, VP of product management at VisualCV, job seekers can also add audio, video clips and links to Word, PDF, and PowerPoint files. VisualCV is using Intridea Inc's MediaPlug on-demand service, which allows developers to use Amazon Web Services (AWS) to transcode and store complex media files.

"The reason we built it with Ruby on Rails is we wanted to have the very strong Ajax interaction," Herman said. "It's not just that the users create and own all the content. It really is a portal builder for a non-technical user. So we really had to have our act together in terms of how the user interface works with very heavy Ajax."

**3. Amazon Web Services Premium Support offering technical assistance for customers of S3, EC2 and SQS:**

New support program from Amazon is offering technical assistance for customers of Amazon Simple Storage Service (**S3**), Amazon Elastic Compute Cloud (**EC2**) and Amazon Simple Queue Service (**SQS**).

Amazon's Silver premium support plan is offering businesses a response time of four hours to two business days and is also providing access to an unlimited number of support cases.

The Gold premium support plan is offering Silver Premium Support Plan along with around-the-clock phone support and one-hour response time for urgent matters.

The customer when signing up to the account can select AWS Premium Support.

**4. SOA for pets also not just for humans:**

Service-oriented architecture (SOA) is not just for humans any more. VetSource Inc. has developed an SOA e-commerce application that provides veterinary prescriptions at home.

According to Craig Sutter, the technical director at VetSource, 'With the new service veterinary hospitals can set up their own website where pet owners can order medications directly once they have a prescription from their veterinarian'.
RESTful e-commerce applications (in Java) use MuleSource Inc's ESB (enterprise service bus) technology for the information from a MySQL. The SOA implementation also uses the Apache Web Server. VetSource also has an Ajax-based dashboard that provides sales and other business data.

Ajax is used not only for interactive user interface but also to provide real-time data.

The VetSource SOA is based on RESTful services, but can interact with SOAP-based Web services. According to Sutter, "All the services we're deploying in Mule are RESTful services. We use SOAP externally. Some of our partners provide their services over SOAP."

**5. Microsoft contributes Web services protocol:**

Microsoft has announced its contribution in Web services protocol specification for consumer scanning peripherals (WS-Scan) to the Printer Working Group (PWG). This will help to make the products interoperable across multiple platforms. It will also provide new functionality and manageability improvements to customers. WS-Scan is a common framework for sharing information between Windows Vista and consumer scanning peripherals.

Jerry Thrasher, PWG chair and senior standards engineer for Lexmark International Inc. said, "Microsoft's WS-Scan specification is a significant contribution to the Printer Working Group. It will greatly help us in our effort for industry wide standardization of networked multifunction device behaviors and capability representation. The PWG Semantic Model, a widely adopted model of network printer behaviors and capabilities, is being extended to include multifunction devices. Maintaining a consistent model for behaviors and capabilities of multifunction devices within the industry not only improves interoperability across operating environments, but also helps reduce implementation costs for device manufacturers."

# Automate data entry with Web services and Ajax

Deploy Web 2.0 technologies to save time and ensure data accuracy

**Level: Advanced**

Norbert (Norb) R. Ryan, III (nryan@us.ibm.com), Software Engineer, IBM
14 Feb 2008

Let's cut through the chatter and find out how a Web service and Asynchronous JavaScript + XML (Ajax) can improve an application, in this case a Ruby on Rails (RoR) application. This article shows you how to spruce up a common Web activity—entering a street address—with Ajax and a call to a Web service. Learn a few tricks to combining these fundamental Web 2.0 components.

**A little background on the idea**
The United States Postal Service (USPS) has made several Web services available (see the USPS Web tools sidebar). One of those Web services accepts a ZIP code and returns the corresponding city and state. In this example application, you make use of this CityStateLookupRequest to save the user some typing. This feature also gives you better address data in your database, because you reduce the chances of typing errors.

**Prerequisites and assumptions**
David Heinemeier Hansson, the ideological spirit and creator of Ruby on Rails, is a smart guy! In RoR, he implemented a number of great ideas, ideas that make developing Web applications much easier and, as a friend of mine remarked, "It makes programming fun again!" There's little doubt in my mind that other frameworks and programming paradigms will espouse these ideas. However, this isn't a tutorial on how to create an RoR application. (See the Resources section at the end of this article for links to good tutorials and reference information.)

The assumption here is that you've created an RoR application that has an HTML input form for an address (for example, 590 Madison Ave, New York, NY 10022). This Rails application also has a model named *address* and a corresponding database table.

**Furthermore, let's assume that you:**

- Understand basic design principles of Web application development.
- Have created an RoR application.
- Understand the basic parts of an RoR application: ActiveSupport, ActiveRecord, ActionView, ActionController, Migrations, and so on.
- Have a database (like IBM® DB2® or MySQL) configured to work with the RoR application.
- Connect with users by anticipating their needs and know the importance of saving them time.

**Table 1. Assuming you have an RoR application with these objects**

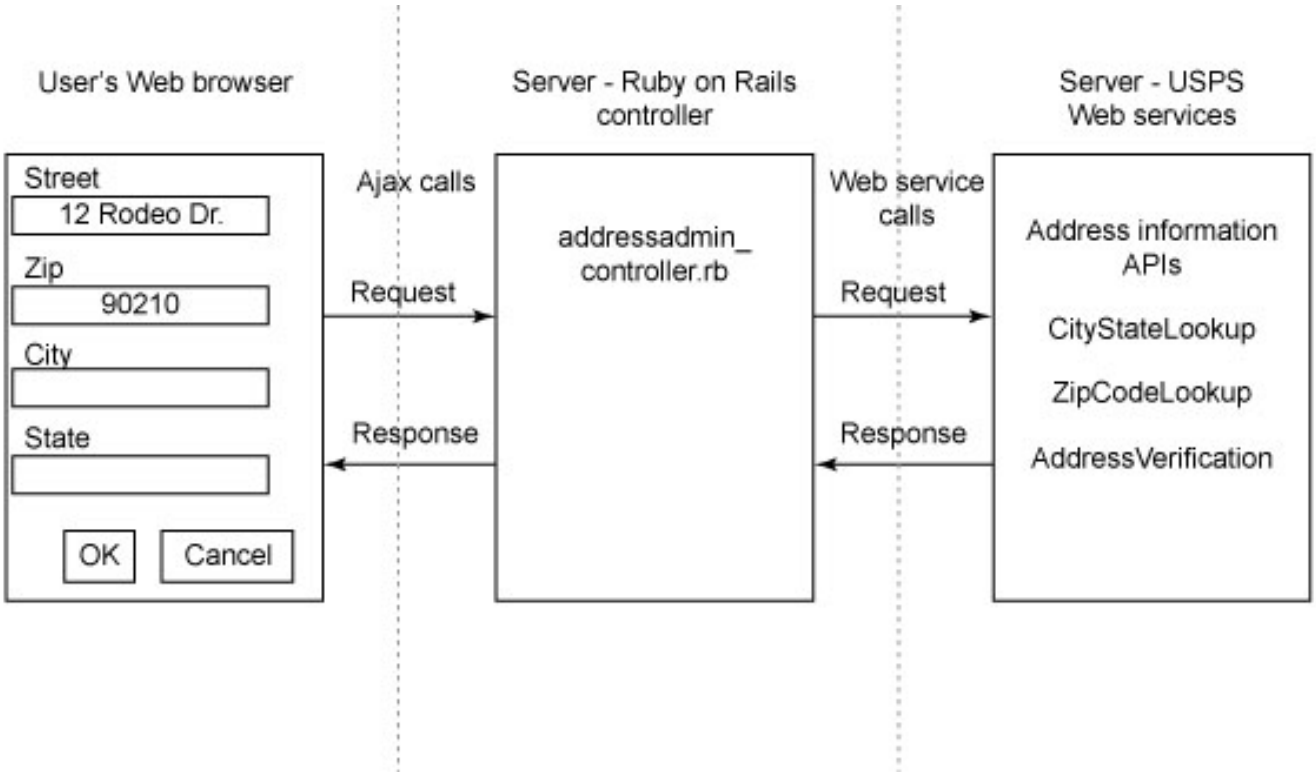
| | | |
|---|---|---|
| edit.rhtml | ../app/views/addressadmin | View for editing an address |
| _form.rhtml | ../app/views/addressadmin | Partial used by edit.rhtml |
| addressadmin_controller.rb | ../app/ | controllersController called by the HTML input form |
| address.rb | ../app/models | ActiveRecord object |
| 001_create_addresses.rb | ../db/migrate | Script for creating the *Addresses* database table |

# Automate data entry with Web services and Ajax

**Solution overview**

The list below shows the steps involved in completing this solution. (Don't worry; the remainder of this article walks you through these steps one by one.) Note that a *partial* is a Ruby on Rails term. It's a reusable piece of code related to what gets displayed in the Web browser. Most modern frameworks include some kind of template and partial functionality that dynamically assembles pieces of the template to produce a Web page. Partials are a real convenience to application developers and drastically help reduce the burden of development. The RoR naming convention is to prefix a partial with an underscore (for example, _addressForm.rhtml).

- Modify the _form.rhtml partial to display the ZIP code before the city and state.
- Add a partial (_cityState.rhtml) to display the input fields for city and state.
- 3. Modify the _form.rhtml partial to "listen" for changes to the ZIP code field and make an Ajax call to the server.
- Modify the controller to validate the ZIP code (5 numeric digits). If not valid, return a blank Ajax response to the client.
- Modify the controller to create a valid XML request to send to the USPS Web service.
- Modify the controller to receive and parse a XML response from the USPS Web service.
- Modify the Ajax response to populate the _cityState partial with the Web services' values.
- Figure out some ways to improve the solution, and e-mail the author with your suggestions.




**Step 1: Modify the view**

The usability of this address form may be called into question. In the U.S., it's normal for the address input fields to be listed in the order of:

- Street
- City
- State
- ZIP

But the form here lists the fields in this order:

- Street
- ZIP
- City
- State

Let's assume that this usability issue can be overcome by user training, instructional text, or common sense. It might also make sense to darken the city and state fields or to prevent entry in them all together. Work with your usability experts to craft an acceptable solution.
Listing 1 shows the input form named _form.rhtml and the input text fields. Notice that the input text field zip5 has been moved above city and state. The last line, debug(params), is optional. During the development and test phases, I usually include this debug data in my RoR views.
Listing 1. Partial with order of input fields changed (_form.rhtml)

```
<%= error_messages_for 'address' %>

<p><label for="address_street">Street</label><br/>
<%= text_field 'address', 'street'  %></p>
<p><label for="address_zip5">Zip5</label><br/>
<%= text_field 'address', 'zip5', :size => "9", :maxlength => "5"  %></p>

<p><label for="address_city">City</label><br/>
<%= text_field 'address', 'city'  %></p>

<p><label for="address_state">State</label><br/>
<%= text_field 'address', 'state'  %></p>

<%= debug(params) %>
```

**Step 2: Add a Rails partial**

The second step in the solution is to break up the _form.rhtml input form by separating the city and state input fields into a new partial. The RoR naming convention is to prefix an underscore to partials. Therefore, the name of the new partial is _cityState.rhtml. The new file resides in the same directory as _form.rhtml. Listing 2 shows the code for the new file, _cityState.rhtml.

**Listing 2. New RoR partial (_cityState.rhtml)**

```
<p><label for="address_city">City</label><br/>
<%= text_field 'address', 'city'  %></p>
<p><label for="address_state">State</label><br/>
```

```
<%= text_field 'address', 'state'  %></p>
```

It would be nice to be able to leave the city and state in the same file as the other address fields. I tried to get it to work but was only able to make it work this way. Why? The difficulty has to do with updating multiple form fields with the response from the Ajax call. Either my experience level with RoR is insufficient, or the generated JavaScript code can't handle it. Most likely the former. Listing 3 shows the code for the _form.rhtml partial after removing the city and state input fields. Note that the new code is given an id = "ajaxLookup"; this is explained in the next step.

**Listing 3. Including the new partial (_form.rhtml)**

```
<%= error_messages_for 'address' %>

<p><label for="address_street">Street</label><br/>
<%= text_field 'address', 'street'  %></p>

<p><label for="address_zip5">Zip5</label><br/>
<%= text_field 'address', 'zip5', :size => "9", :maxlength => "5"  %></p>

<div id = "ajaxLookup">
  <%= render :partial => "cityStateFields" %>
</div>
```

The naming convention is an important idea in RoR. RoR assumes that partials are named with a leading underscore. And RoR assumes that references to that partial won't contain an underscore. So in Listing 3, it's expected that the line won't have an underscore: <%= render :partial => "cityStateFields" %> is correct. RoR sees this line and looks in the same directory for a file named _cityStateFields.rhtml.

**Step 3: Listen for changes to the ZIP code**

Rails has built-in support for Ajax. This is one of the areas where Rails really shines. Simply add the lines of code shown in Listing 4 to listen for changes to the ZIP code field.

**Listing 4. Add an Ajax Listener to ZIP code (_form.rhtml)**

```
01 <%= javascript_include_tag :defaults %>
02
03 <p><label for="address_street">Street</label><br/>
04 <%= text_field 'address', 'street'  %></p>
05
06 <p><label for="address_zip5">Zip5</label><br/>
07 <%= text_field 'address', 'zip5', :size => "9", :maxlength => "5"  %></p>
08
09 <div id = "ajaxLookup">
10  <%= render :partial => "cityStateFields" %>
11 </div>
12
13 <%= observe_field :address_zip5,
14            :frequency    => 2.00,
15            :update       => "ajaxLookup",
16            :url          => {:action => :cityStateSearch, :id => @address},
17            :with         => "'zip5=' + encodeURIComponent(value)"
```

```
18 %>
19
20 <%= debug(params) %>
```

**Note:** The two-digit line numbers on the very left edge of the listing are for explanation purposes; they don't appear in the code.

That's it! In about 10 lines of Ruby code, this view has been embellished with Ajax functionality. Behind the scenes, RoR and the prototype library handles all the JavaScript. Let's go through these 20 lines of code.

Line 01 instructs RoR to include the Prototype and Scriptaculous JavaScript libraries. Lines 03-12 are the same as before. Line 13 uses the observe_field method from the Prototype library. observe_field is a helper method in the PrototypeHelper class. In plain language, lines 13-17 say to check the zip5 input field every two seconds. If the zip5 input field has user input, then call the action cityStateSearch in the current controller, which is addressadmin_controller.rb. This logic is being run by JavaScript within the user's browser. When the zip5 input field is changed, an Ajax call is made from the user's browser to the server. Notice the correlation between line 09 and line 15: Line 15 identifies what to do with the response from the action cityStateSearch. The response of the action, if any, updates the  tag named ajaxLookup. Line 09 has the div tag with the ID equal to ajaxLookup. So the response from the action cityStateSearch is passed into line 10. Line 17 explains what name and value pair to send to the action. So in this example, the string zip5=90210 is passed to addressadmin_controller's action named cityStateSearch.

Next, you begin work on the controller functionality. The last step specified that the user's browser would make an asynchronous call to the action named cityStateSearch when it detected a change in the ZIP code (that is, zip5) input field. Here are the main pieces of functionality that you have to code on the server side:

- Validate the ZIP code.
- Build the XML to call the Web service.
- Call the Web service.
- Parse the response from the Web service.
- Send the response back to the user's Web browser.

### Step 4: Validate the ZIP code

There's no sense calling the USPS Web service with a invalid ZIP code. With a little effort you can eliminate most invalid ZIP codes. In the U.S., the ZIP code is five numbers. Listing 5 shows some code that checks to see if the zip5 parameter consists of five numbers.

**Listing 5. Validate the ZIP code (addressadmin_controller.rb)**

```
01  def cityStateSearch
02
03    if params[:zip5].nil?
04      logger.debug("zip5 is null")
05    elsif !(params[:zip5] =~ /\d{5}/)
06      logger.debug("We have a bad ZIP code — not 5 digits.")
07      logger.debug("zip5 = #{params[:zip5]}")
08    else
09      logger.debug("We have a good 5-digit ZIP code.")
10      logger.debug("zip5 = #{params[:zip5]}")
```

```
11
12     if params[:address].nil?
13        @address = Address.new
14     else
15        @address = Address.find(params[:id])
16        if @address.update_attributes(params[:address])
17          flash[:notice] = 'Address was successfully updated.'
18        end
19     end
20   end
21
22 end   #cityStateSearch
```

Line 01 starts the definition for the new action cityStateSearch. Unlike other actions in this controller, the cityStateSearch action is called asynchronously by the JavaScript—the client-side Ajax code. Line 03 checks to see if the parameter value is null (or nil, as Ruby calls it). Line 05 is a regular expression that compares the parameter string value against /\d{5}/, which we all know and love as the regular expression for five digits. The exclamation point before the expression negates the elsif expression. (Yes, that's correct RoR syntax for what is otherwise known as *else if*.)

Lines 12-19 handle creating or updating the @address object. There's a little subtlety in lines 5-7. When the logic falls into these lines, it drops out of the action and returns to the browser. The end user is none the wiser. This action skips the rest of the logic and returns right away when the zip5 is nil or is not 5 digits. As long as the user's cursor is in the zip5 input text field on the view, then this action gets reinvoked every two seconds. That functionality was configured earlier in the observe_field method in the frequency parameter.

Line 21 is where you add the next section of code. Because this is development code, I use lots of debug statements. After the code has been polished and refined several times, I'll remove the logger.debug statements. Also, I'm an RoR newbie, so lots of debug statements give comfort to my style of programming, which has been accurately described elsewhere as "beat it into shape."

**Step 5: Create a valid XML request to send to the USPS Web service**

At this point in the process, you're in the server side of your RoR application, and you have a ZIP code with five digits. Because you have a reasonable expectation that the ZIP code is legitimate, it's now worth the effort to call the USPS Web service. To do this, you need to create a valid request. Jumping ahead a little, Listing 6 shows an example of a valid XML request.

**Listing 6. Valid XML request**

```
http://testing.shippingapis.com/ShippingAPITest.dll?API=CityStateLookup
&XML=<CityStateLookupRequest%20USERID="XXXXXXXXXXXX"><ZipCode ID=
"0"><Zip5>90210</Zip5></ZipCode></CityStateLookupRequest>
```

To use the USPS Web Tools, you have to register with them. Registration is easy and free (see the Resources section for more information). After I registered for the USPS Web Tools, they sent me the name of the test server and a user ID. (In Listing 6 I did my best impersonation of a top-secret government agent by crossing out my user ID with XXXXXXXXXXXX.) Let's parse this request a little more. The Web service endpoint is specified by API=CityStateLookup. In the HTML form, you can now see why I called the input field zip5. That's the name that the USPS request expects. This CityStateLookup Web service accepts up to five ZIP code values in one request. To keep things

simple, this code only passes one ZIP code, which has the XML tag <ZipCode ID= "0">. So the functionality becomes quite obvious: You need to get the five-digit value entered by the user and put it into this XML tag named <Zip5>.

**So what kind of Web service is this?**

It turns out that this seemingly simple question isn't always easy to answer. Web services have become like Baskin Robbins' 31 flavors of ice cream or the Starbucks coffee menu. It seems like ordering a cup of coffee should be easy to do until they hit you with something called a Grande Chai Latte with Soy. First, what it's not: This isn't a XML-RPC-style Web service, document-style Web service, SOAP Web service, or Representational State Transfer (REST) (noun-based) request Web service. The designers at the USPS have decided to implement this as a plain vanilla XML Web service. The USPS Web servers accept either GET or POST HTTP requests. The requests are stateless with no cookies or URL rewrites. Requests and responses are case sensitive. Once again, it's easy to register with USPS Web Tools, and there's plenty of documentation. (I should note that I have no affiliation with the USPS.)

To create the XML, you use the Builder::XmlMarkup library, which is included with RoR. At the beginning of the controller class file addressadmin_controller.rb, you need to add the code shown in Listing 7.

**Listing 7. Code to be added to addressadmin_controller.rb**

```
require 'open-uri'
require 'uri'
require 'rubygems'
require_gem 'builder'
require "rexml/document"
```

**Listing 8. Create the XML portion of the request**

```
01  def cityStateSearch
02
03    if params[:zip5].nil?
04      logger.debug("zip5 is null")
05    elsif !(params[:zip5] =~ /\d{5}/)
06      logger.debug("We have a bad ZIP code — not 5 digits.")
07      logger.debug("zip5 = #{params[:zip5]}")
08    else
09      logger.debug("We have a good 5-digit ZIP code.")
10      logger.debug("zip5 = #{params[:zip5]}")
11      #  Build the XML to call the web service
12      xm = Builder::XmlMarkup.new
13       xmlstuff = xm.CityStateLookupRequest("USERID"=>"XXXXXXXXXXXX") {
14        xm.ZipCode("ID"=>"0") {
15          xm.Zip5(params[:zip5]) }}
16
17    end
18  end   #cityStateSearch
```

Just four lines, lines 12 through 15, create the properly formatted XML for the request. The string variable xmlstuff contains this XML:

```
<CityStateLookupRequest%20USERID="XXXXXXXXXXXX"><ZipCode ID= "0"><Zip5>90210</
```

```
Zip5></ZipCode></CityStateLookupRequest>
```

There are a couple of more important steps to getting this request properly formatted. You need to escape the special characters of the request with the two lines of code in Listing 9.

**Listing 9. Escape the request's special characters**

```
uri_enc = URI.escape(‘http://testing.shippingapis.com/ShippingAPITest.dll
    ?API=CityStateLookup&XML=’ + xmlstuff)
uri = URI.parse(uri_enc)
```

These lines take care of the conversion of all the special characters into proper encoding as an HTTP request. I'll leave it up to you to improve this code by setting up variables or property files for the server name, API name, and so on. Your goal is just to get the call to the Web service working; you can polish and refine later. Now you have a properly formatted HTTP/XML request and are ready to invoke the USPS Web service.

**Step 6: Call the Web service and receive a response**

In this step, you call the USPS Web service and receive a response. The code must parse the XML response. Listing 10 shows a sample of a USPS CityStateLookup response.

**Listing 10. USPS CityStateLookup response**

```
<?xml version=”1.0"?>
<CityStateLookupResponse><ZipCode ID=”0"><Zip5>90210</Zip5>
<City>BEVERLY HILLS</City><State>CA</State></ZipCode>
</CityStateLookupResponse>
```

Again, I took this example directly from the USPS Web Tools documentation. Your goal in this step is to parse the city and state information and place it in your @address object's variables. Eventually those variables are returned as a part of the Ajax response.
**Keep this in mind:** the Builder library allows for creation of the XML, while the module REXML enables parsing the XML data.

**Listing 11. Call the Web Service and parse the response (addressadmin_controller.rb)**

```
# The call to the Web service — response is in var ‘doc’
 doc = REXML::Document.new open(uri)
 logger.debug(“doc = “ + doc.to_s)
 doc.elements.each(“CityStateLookupResponse/ZipCode”) { |element|
  logger.debug(element)
  logger.debug(“element[0] = “ + element[0].to_s)
  logger.debug(“element[0].text = “ + element[0].text)
  logger.debug(“element[1] = “ + element[1].to_s)
  logger.debug(“element[1].text = “ + element[1].text)
  logger.debug(“element[2] = “ + element[2].to_s)
  logger.debug(“element[2].text = “ + element[2].text)
   # Set the model field values to the response from the Web service
   @address.city = element[1].text
   @address.state = element[2].text
 }
```

The code in Listing 11 iterates through the XML response to find the city (element[1]) and the state (element[2]). element[0] is the zip5 value. As mentioned earlier, the Web service processes up to five ZIP code lookups in one request. A future refinement of this code should consider looping through those values. But in this simple example, you're always only passing in one ZIP code value per request. This code could also do a better job of handling no city/state response values. The point here is to get minimal function working so you can refine it per the constraints or demands of your particular project. If you have questions about the use of the Builder library or the REXML module, see the RoR API documentation., which has plenty of examples.

If you're familiar with the way other languages create or parse XML, you'll immediately recognize how easy it is in RoR. One of the messages that RoR is quietly shouting is, *It doesn't have to be a huge pain in the butt!* XML creation—no sweat. Ajax—with one hand behind my back. XML parsing—easy peasy. RoR frequently reminds me that the objective is to get real work done, real quick, for real end users. It's a nice feeling.

Recapping the action, you validated the ZIP code, created the XML request to call the Web service, parsed the XML response, and put the city and state fields in the @address object. From the point of view of the user, all of this processing has been done asynchronously and in about one second. The user's cursor is still in the ZIP code field on the html form of the Web browser. Before they know it, this action is returning the @address object with the correct city and state values. And the information is sent to the partial _cityStateFields.rhtml and populated in the user's browser. Hopefully it's a good surprise to the user—and you've given the feeling that you're on their side and are here to help them out. You've anticipated their needs and done your best to make their lives a little easier.
Listing 12 shows the full code for the cityStateLookup action in the file addressadmin_controller.rb.

**Listing 12. Full code for action cityStateLookup (addressadmin_controller.rb)**

```
require 'open-uri'
require 'uri'
require 'rubygems'
require_gem 'builder'
require "rexml/document"

class AddressadminController < ApplicationController

  <!— other methods/actions —>

  def cityStateSearch

    if params[:zip5].nil?
      logger.debug("zip5 is null")
    elsif !(params[:zip5] =~ /\d{5}/)
      logger.debug("We have a bad ZIP code — not 5 digits.")
      logger.debug("zip5 = #{params[:zip5]}")
    else
      logger.debug("We have a good 5-digit ZIP code.")
      logger.debug("zip5 = #{params[:zip5]}")

      if params[:address].nil?
        @address = Address.new
      else
```

```
      @address = Address.find(params[:id])
      if @address.update_attributes(params[:address])
        flash[:notice] = 'Address was successfully updated.'
      end
    end

    #  Build the XML to call the Web service
    xm = Builder::XmlMarkup.new
     xmlstuff = xm.CityStateLookupRequest("USERID"=>"XXXXXXXXXXXX") {
    xm.ZipCode("ID"=>"0") {
    xm.Zip5(params[:zip5]) }}

    webservice = 'http://testing.shippingapis.com/ShippingAPITest.dll?'
    uri_enc = URI.escape(webservice + 'API=CityStateLookup&XML=' + xmlstuff)
    uri = URI.parse(uri_enc)

    # The call to the Web service — response is in var 'doc'
    doc = REXML::Document.new open(uri)
    logger.debug("doc = " + doc.to_s)
    doc.elements.each("CityStateLookupResponse/ZipCode") { |element|
     #logger.debug(element.attributes["name"])
      logger.debug(element)
     logger.debug("element[0] = " + element[0].to_s)
     logger.debug("element[0].text = " + element[0].text)
     logger.debug("element[1] = " + element[1].to_s)
     logger.debug("element[1].text = " + element[1].text)
     logger.debug("element[2] = " + element[2].to_s)
     logger.debug("element[2].text = " + element[2].text)
      # Set the model field values to the response from the web service
      @address.city = element[1].text
      @address.state = element[2].text
    }
  end  # valid ZIP code if-statement-checkers
 render :partial => "cityStateFields"
end
```

**Miscellaneous thoughts**

Some miscellaneous comments on this solution and Web services:

- Web services work regardless of the programming language. This example used Ruby on Rails, but it also works with other languages and frameworks.
- This solution lacks a plan for what to do when the USPS Web service isn't available. Maybe a local cache can help minimize the impact of an outage.
- The parsing of the XML response from the Web service is probably the least elegant code written. Maybe a template can help this area of the solution.
- Why bother the user with entering a city and state when it's possible to look them up with just a ZIP code?
- How effective are these various *working groups* that bequeath Web service standards into the marketplace?
- Has anybody ever thought for one second about implementing the Universal Description

Discovery Integration (UDDI) standard?
- Are Web service standards too complex? Can developers wade through all the acronyms?
- Rarely does the marketplace declare only one winner. Many competitors advance to the next round. But the marketplace seems quite efficient at weeding out the losers.
- How long until the naysayers question RoR's performance, security, or production worthiness?

The marketing muscle of many big companies is clouding the Web service waters, and this article addressed a common problem with a simple and easily understood Web service. The people at the United States Postal Service have implemented a solid and very useful Web Service. Jakarta Struts was a vast improvement over previous efforts to address Web application frameworks—to address the whole model view controller (MVC) stack. Ruby on Rails is, at least, a significant improvement over Struts.

# Implementing WS-Security

**Implementing WS-Security**
A case study

**Level:** Introductory
Sam Thompson, jStart Program Manager for Web services, IBM

01 Apr 2003

This article describes how the emerging WS-Security standard was used to secure a Web service that was developed and deployed in the fall of 2002. The article will discuss the security-related requirements of the Web service and how they were met using a combination of HTTPS/SSL, digital certificates, and digital signature technologies. The article will crawl through the WS-Security element of the SOAP message used to trigger the Web service, explaining each section of the WS-Security element in detail. After reading this article, you should have a better understanding about how you can use WS-Security in a production Web service application and feel confident about using this emerging standard in your own projects.

**Web services security - ready to do business**
Over the past couple of years, Web services has gone from being an overly-hyped technology to one that many organizations are using productively. The early implementations, like all new technology projects, tended to be sandbox-type efforts or projects that were small, inside the firewall, and non-mission-critical in nature. Those brave souls that tried to venture into the world of delivering Web services over the Internet found that they either had to provide services that were open and available for use by anyone (for example XMethods or Amazon) or had to develop their own, typically proprietary, very company-specific, security scheme.

Early adopters using the Internet as their transport typically used some form of registration process (for example Google) for open Internet services or only provided services to a small number of business partners with whom they already had a tight, trusted relationship. For example, in order to use Google's Web service-enabled search engine, the service requester must first register with Google through an HTML based form. As part of the registration process, Google sends the requester an email with a security "token". When the requester invokes the service, they provide this token to Google as part of the SOAP message to verify that they are a registered, authorized user of the Google Web service.

In these situations, even though service providers were using industry standards such as SOAP, additional information concerning the security scheme/process needed to be provided in order for the service requestors to be able to use the service. This had a rather undesired effect of tightly coupling the requester and the provider, a scenario that wasn't desired by either party.

**The WS-Security Standard**
Clearly an industry standard way of securing a Web service was required, and IBM, Microsoft, and VeriSign responded to this need in April, 2002. From the WS-Security specification (see also Resources):

"WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.
WS-Security also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible (e.g. support multiple security token formats). For example, a client might provide proof of identity and proof that they have a particular business certification."

# Implementing WS-Security

Since 1997, IBM has had a program called jStart (short for jump-start — see Resources) to help its customers and business partners work with new emerging technologies. The program's goal is to help early adopters leverage new technologies to help make their businesses more successful. Last fall, the jStart program worked with a company who wanted to provide a business-to-business Web service using the Internet as a transport. They desired a strong level of security and interoperability, and they decided to use a WS-Security approach to secure the SOAP message traffic with their business partners. This paper discusses that project and its use of WS-Security.

**Why WS-Security?**
As the use cases for our customer's application were being developed, a set of security-related, non-functional requirements were identified:

1. The communication between our customer and his business partner should not be able to be viewed by a third party as it travels on the Internet.
2. Our customer needed to be able to determine from whom the message was coming and be able to verify that the sender was who the sender claimed to be.
3. Our customer needed to be able to ensure that the data being transmitted was not tampered with.

Non-functional requirement #1 will be addressed through the use of HTTPS/SSL transport security. Since this application will be a point-to-point application, with no third party service providers or intermediaries involved, the idea of using cryptography to encrypt all or part of the SOAP message was evaluated but not implemented at this time. Given no third parties were involved, the value gained from an additional encryption step that would encrypt a segment of the SOAP message was not enough to justify the additional development expense and complexity that would have been needed to implement a form of message-level encryption.

Non-functional requirements #2 and #3 will be addressed through the use of digital signatures and digital certificates. When using a digital certificate approach, the Web service requester must have a digital certificate which has been signed by a *trusted certificate authority*. The requester will use this certificate to assert their identity and will digitally sign the SOAP message so that the requester's identity and the message's integrity can be verified.

Once the message is received at our customer's system, it will be time stamped and logged. At this point, the digital signature will be validated. The validation process will ensure that the message came from the sender as well as verify that the message contents have not been modified since it was signed at the sender's site. The SOAP message log that our customer creates in DB2 will be used for non-repudiation purposes.

**The Web service**
Now that you understand the requirements and the technical approach, let's take a look at what was implemented. The application that our customer chose to implement as a Web service was developed using WebSphere Studio Application Developer and some tools from the IBM alphaWorks Web site, namely, the XML Security Suite, and the Apache Axis run time that was part of the IBM Web Services Toolkit. Although the application is quite powerful as it drives our customer's core business application, it is simple in that it only implements one method. It was deployed on WebSphere Application Server and interacts with the customer's core business application through WebSphere

**MQ Series.**
By using the TCP/IP monitor that is part of Application Developer, we've captured the SOAP message that is sent to the Web service for processing. Note that in order to maintain the confidentiality of our customer, we made the SOAP URLs generic, removed the application-specific SOAP payload,

and slightly modified some of the calculated values:

```
1.  <soapenv:Envelope 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”
        xmlns:SOAP-ENC=”http://schemas.xmlsoap.org/soap/encoding/”>

2.  <soapenv:Header>

3.  <wsse:Security soapenv:actor=”http://www.jStartcustomer.com/actors#verifier”
        soapenv:mustUnderstand=”1"
        xmlns:wsse=”http://schemas.xmlsoap.org/ws/2002/04/secext”>
       <Signature xmlns=”http://www.w3.org/2000/09/xmldsig#”>

4.  <SignedInfo>

5.  <CanonicalizationMethod Algorithm=”http://www.w3.org/2001/10/xml-exc-c14n#”/>

6.  <SignatureMethod Algorithm=”http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>

7.  <Reference URI=”#sign_content_1043176028580">

8.   <Transforms>

9.  <Transform Algorithm=”http://www.w3.org/2001/10/xml-exc-c14n#”/>

10.   </Transforms>

11.  <DigestMethod Algorithm=”http://www.w3.org/2000/09/xmldsig#sha1"/>

12.  <DigestValue>FLuQTa/LqDIZ5F2JSaMRHSRuaiQ=</DigestValue>

13.  </Reference>

14.  </SignedInfo>

15.  <SignatureValue>

16.  kGlrrXjKku/WXKxID+JJkEXY+aGNYHc5dy8GwbLFtB5Msll2/MhwdnO9wastJ0gLPzLy3oHL

17.   7A8ggkMkjgAqnLg6PTzM7MdKoIAhe+xRHdOysamGucFJQRMrU+JQ4WATJt0bpdClwJy6mexT

18.   Su48mq1q5rM9YZh61P7UEUKt+EQ=

19.  </SignatureValue>

20.  <KeyInfo xmlns=”http://www.w3.org/2000/09/xmldsig#”>

21.  <KeyValue>

22.   <RSAKeyValue>
```

```
23.  <Modulus>
24.   2sW+eBjx5D2QMyr8ocZIZWNYHGf9zYhB4XWILPCTvhNV7dIe3l8ARepOA1ABFK2OMy
25.  pzb+Rb+nWQeo//yFz/28PmL63kdLiE72qmmQuzuPa5NXaV9pJ4JKw86QdLhGGpFIRH
26.   18Iugf3xLFwQEZqKYnblTUs7ftnTgW5r4HH492k=
27.  </Modulus>
28.   <Exponent>AQAB</Exponent>
29.  </RSAKeyValue>
30.  </KeyValue>
31.  <X509Data>
32.  <X509IssuerSerial>
33.  <X509IssuerName>OU=Java,O=IBM,L=Unknown,ST=Oklahoma,C=US</X509IssuerName>
34.  <X509SerialNumber>0</X509SerialNumber></X509IssuerSerial>
35.  <X509SubjectName>CN=John Doe</X509SubjectName>
36.  <X509Certificate>
37.
MIIB0TCCAToCAQAwDQYJKoZIhvcNAQEEBQAwTzELMAkGA1UEBhMCVVMxETAPBgNVBAgTCE9rbGFo
38.
b21hMRAwDgYDVQQHEwdVbmsam3duMQwwCgYDVQQKEwNJQk0xDTALBgNVBAsTBEphdmEwHhcNMDIw
39.
OTI1MTAxMTQ4WhcNMDMwOTI1MTAxMTQ4WjATMREwDwYDVQQDEwhKb2huIERvZTCBnzANBgkqhkiG
40.
9w0BAQEFAAOBjQAwgYkCgYEA2sW+eBjx5D2QMyr8ocZIZWNYHGf9zYhB4XWILPCTvhNV7dIe3l8A
41.   RepOA1ABFK2OMypzb+Rb+nWQeo//yFz/
28PmL63kdLiE72qmmQuzuPa5NXaV9pJ4JKw86QdLhGGp
42.
FIRH18Iugf3xLFwQEZqKYnblTUs7ftnTgW5r4HH492kCAwEAATANBgkqhkiG9w0BAQQFAAOBgQCs
43.   OD02WMoYcMR7Sqdb9oQyk7Nn4rQ5DBgZ5mxGGVzWxBZW/
QON+Ir2j4KUjX1jalMvbHa9lnhPQmJi
44.  Ued923rza7fvdRG2CDalbW0R3aPd5q0u3akP0/
Ejb7z5o88heajCSgfRruvU+ZdOTT3Oe+RBQgw8
```

```
45.  VuzbLApPnXiehowYuA==
46. </X509Certificate>
47. </X509Data>
48. </KeyInfo>
49. </Signature>
50. </wsse:Security>
51. </soapenv:Header>
52.  <soapenv:Body>
53. application specific data/content
54.  </soapenv:Body>
55. </soapenv:Envelope>:
```

Let's look at the SOAP message in greater detail. As you can plainly see, this is a typical SOAP message with an outermost opening and closing <soapenv:Envelope> tag set. The SOAP envelope contains <soapenv:Header> and <soapenv:Body> sections. The WS-Security section, as defined by the WS-Security specification, is positioned within the SOAP Header and is designated by the opening and closing <wsse:Security> block, lines 3-51. The <Security> header block provides a mechanism for attaching security-related information targeted at a specific receiver (the SOAP actor). Since only one SOAP actor is involved in this use case, only one <Security> header block is contained in the message.

In line 3, the SOAP actor attribute defines the recipient of a header entry, Security soapenv:actor="http://www.jStartcustomer.com/actors#verifier". Line 3 also contains the soapenv:mustUnderstand="1" attribute. By setting the SOAP mustUnderstand attribute to "1", we indicate that the service provider must process the SOAP header entry. As per the SOAP specification, since the attribute is set to "1", if the receiver cannot obey the semantics (as conveyed by the fully qualified name of the element) and process the message according to those semantics, the receiver MUST fail processing the message and generate a fault.

**SignedInfo and Digests**

Lines 4-14, <SignedInfo> </SignedInfo>, describes the signed content of the message. Note that as is customary with digital signature applications, a *digest* is used to facilitate faster processing. This is a standard industry practice and is done for performance reasons. The *payload* (the SOAP Body) of our SOAP message is quite long, and the process of applying a public key algorithm to the full message could significantly impact the performance of our Web service. As such, a *digest* is used. A digest is a fixed length, short message whose digital signature can be quickly generated and verified. When the message is received, our Web service digital signature verifier class (implemented as an Apache Axis plugable provider) will compute the digest and verify that the newly-computed digest matches the digest that was sent.

Let's look at the elements that make up the Signed Content portion of the message. Line 5, <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>, identifies the canonicalization algorithm that is used to create a canonicalized form of the information being signed — in this case, the digest. This step is needed because of the nature of XML documents and the programming tools that work with them. XML documents, in some cases, can have slight textual differences, yet be essentially the same logical document. Small variations in the way comments are represented or in the way an XML parser handles line delimiters when serializing/deserializing an XML data structure can create slightly different binary representations of the same content. If the algorithm that verifies the digital signature were to be run against a slightly different serialized version of the data, the result could be a *fail* when indeed it should be a *pass*.

To avoid this problem, the document is first transformed into its canonicalized form through the use of a canonicalization algorithm. This algorithm, an implementation of the W3C Exclusive XML Canonicalization Version 1.0 Specification (see Resources), a W3C recommendation, transforms the document into its basic canonicalized form. This allows us to get a consistent binary representation that can be correctly compared and thus yield the correct result.

Line 6, <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>, indicates the Signature Method Algorithm. This is the algorithm that is used to convert the output of the canonicalization algorithm into the Signature Value. Our signature algorithm is a combination of a key dependent algorithm (RSA) and a hash algorithm (SHA1). This algorithm is an implementation of the RSASSA-PKCS1-v1_5 specification described in W3C RFC 2437 (see Resources).

Line 7, <Reference URI="#sign_content_1043176028580">, indicates the reference element. The optional URI attribute of Reference identifies the data object that was signed. The Reference block includes the algorithm that is used to compute the digest, the digest value that was computed, and the final transform that is performed prior to computing the digest value. Lines 8-10, <Transforms> <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/> </Transforms>, indicate the transformation algorithm, while lines 11 and 12 specify the digest algorithm and the computed digest value, <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/> <DigestValue>FLuQTa/LqDIZ5F2JSaMRHSRuaiQ=</DigestValue>.

In our application, the Transform algorithm is once again the W3C Exclusive XML Canonicalization algorithm discussed above. The method used to compute the digest, the Secure Hash Algorithm, is part of the U.S. Department of Commerce/National Institute of Standards and Technology's Secure Hash standard.

Lines 15-16, <SignatureValue>kGlrrXjKku/WXKxID+JJkEXY+aGNYHc5dy8GwbLFtB5Msll2/MhwdnO9wastJ0gLPzLy3oHL
7A8ggkMkjgAqnLg6PTzM7MdKoIAhe+xRHdOysamGucFJQRMrU+JQ4WATJt0bpdClwJy6mexT
Su48mq1q5rM9YZh61P7UEUKt+EQ=</SignatureValue>, contain the signature value, which is actually the encrypted digest value. This value is the output of the Signature Method Algorithm indicated on line 6.

**Keys**

Lines 20-48 introduce the concept of keys. A key is used to mathematically transform a normal, readable text message into an unreadable one for transmission across the internet. Our Web service will use a public/private-key (a pair of mathematically related keys) or an asymmetric key encryption scheme. One of these keys is kept secret; this is the private key. In our application, the Web service requester will sign the digest with his private key prior to sending the document

to the service provider.

Line 20 begins the Key Value block and identifies the namespace that we will use — <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">. The Key Value block, <KeyValue> <RSAKeyValue> </RSAKeyValue> </KeyValue>, is where the Web service requester supplies the Web service provider with the information they need in order to obtain the key needed to validate the signature. In our message, the KeyValue element contains the requester's public key, which will be used to validate the signature. We have chosen to use an asymmetric RSA (named after the three inventors, Rivest, Shamir, and Adleman) key-based approach in order to meet our non-repudiation requirement.

The use of the RSA key scheme assures our Web service provider that the message was in the same form that it was received, and it was signed by the owner of the extracted digital certificate. If the recomputed digest matches the decrypted digest from the SOAP message, the service provider is able to verify the integrity of the message. By logging the message before any processing is done (the log is implemented as the first Apache Axis handler in the handler chain), the Web service provider can prove that the message in question was sent by whomever signed the message and that it was received unmodified from the form in which is was sent.

RSAKeyValue elements have two fields:

**Modulus**

```
<Modulus>
2sW+eBjx5D2QMyr8ocZIZWNYHGf9zYhB4XWILPCTvhNV7dIe3l8ARepOA1ABFK2OMy
pzb+Rb+nWQeo//yFz/28PmL63kdLiE72qmmQuzuPa5NXaV9pJ4JKw86QdLhGGpFIRH
18Iugf3xLFwQEZqKYnblTUs7ftnTgW5r4HH492k=</Modulus>
```

**Exponent**

```
<Exponent>AQAB</Exponent>
```

The RSA scheme uses large prime numbers to construct the key pairs. The Modulus is the product of two large prime numbers. Each pair shares the Modulus, but each also has a specific exponent. The RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1 document (see Resources) describes how the Modulus and the Exponent are created:

"Take two large primes, p and q, and compute their product n = pq; n is the modulus. Choose a number, e, less than n and relatively prime to (p - 1)(q - 1), which means that e and (p - 1)(q - 1) have no common factors except 1. Find another number d such that (ed - 1) is divisible by (p - 1)(q - 1). The values e and d are called the public and private exponents, respectively. The public key is the pair (n, e); the private key is (n, d)."

The service requester digitally signs the message with their private key. On the service provider side, the signature is verified using the requester's public key. Since the service requester signs the message with a private, asymmetric key, the service provider is assured that the only organization that could have signed the message is the holder of the private key.

**Digital Certificates**

The next section of the Key Info block is the digital certificate itself as indicated by the <X509Data> element. The digital certificate is used to identify the sender of the message in the way a user ID is used to identify the user of Web and enterprise applications. The first element in this block of data identifies the organization that signed the certificate. This is typically the certificate authority. In our

case, that information has been replaced with generic information:

<X509IssuerSerial> <X509IssuerName>OU=Java,O=IBM,L=Unknown,ST=Oklahoma,C=US</X509IssuerName> <X509SerialNumber>0</X509SerialNumber></X509IssuerSerial>

Next is the <X509SubjectName>CN=John Doe</X509SubjectName> element, which contains the distinguished name of the service requester — in our case, simply John Doe — and finally the X.509 certificate itself:

```
<X509Certificate>
MIIB0TCCAToCAQAwDQYJKoZIhvcNAQEEBQAwTzELMAkGA1UEBhMCVVMxETAPBgNVBAgTCE9rbGFo
b21hMRAwDgYDVQQHEwdVbmsam3duMQwwCgYDVQQKEwNJQk0xDTALBgNVBAsTBEphdmEwHhcNMDIw
OTI1MTAxMTQ4WhcNMDMwOTI1MTAxMTQ4WjATMREwDwYDVQQDEwhKb2huIERvZTCBnzANBgkqhkiG
9w0BAQEFAAOBjQAwgYkCgYEA2sW+eBjx5D2QMyr8ocZIZWNYHGf9zYhB4XWILPCTvhNV7dIe3l8A
RepOA1ABFK2OMypzb+Rb+nWQeo//yFz/
28PmL63kdLiE72qmmQuzuPa5NXaV9pJ4JKw86QdLhGGp
FIRH18Iugf3xLFwQEZqKYnblTUs7ftnTgW5r4HH492kCAwEAATANBgkqhkiG9w0BAQQFAAOBgQCs
OD02WMoYcMR7Sqdb9oQyk7Nn4rQ5DBgZ5mxGGVzWxBZW/
QON+Ir2j4KUjX1jalMvbHa9lnhPQmJi
Ued923rza7fvdRG2CDaIbW0R3aPd5q0u3akP0/Ejb7z5o88heajCSgfRruvU+ZdOTT3Oe+RBQgw8
VuzbLApPnXiehowYuA==
</X509Certificate>
```

The final stage of WS-Security processing is to validate the Web service requester's digital certificate. When using a digital certificate approach, each Web service requester must have a digital certificate which a trusted Certificate Authority (CA) has signed. The definition of a trusted certificate authority is outside the scope of this paper, but typically either an industry-accepted, 3rd-party certificate authority (a company like VeriSign) performs this role, or the Web service provider takes on the role of the certificate authority for the certificates that their application uses. If the latter approach is used, use of the open source OpenSSL toolkit from Apache, or the basic digital certificate support provided by WebSphere Application Server's IKEYMAN utility is recommended.

In the initial rollout of our application, our customer chose to play the role of certificate authority. As such, they created their own, self-signed CA certificate. Prior to any Web service interactions, the Web service requester must provide the Web service provider with a certificate that they will use in the application. Playing the certificate authority role, the Web service provider signs the certificate and returns it to the Web service requester.

As described above, the Web service requester includes the CA-signed digital certificate in the SOAP message. After the digital signature has verified the integrity of the message, the certificate is extracted from the message and validated using the CA's public key. Once the validity of the certificate has been achieved, the Web service requester will have been authenticated and the processing of the WS-Security portion of the message will be complete. The owner has successfully authenticated to the receiver.

Note that if, at a later date, our service provider chooses to relinquish the role of the certificate authority and use an industry-accepted, trusted, 3rd-party certificate authority, the logic of our validation code does not need to change. In this scenario, prior to using the Web service, the Web service requester will need to get a certificate issued by a trusted certificate authority. The service requester will place this 3rd-party certificate in the SOAP message. When the service provider is validating the digital certificate, instead of using their self-signed CA key, the public key of the 3rd-party CA will be used. In this scenario, the service provider has established a trust relationship with

the 3rd-party certificate authority, and the CA will be trusted to adequately authenticate users before issuing certificates to them.

**WS-Security and WSDL**

One of the promises of Web services is to be able to loosely couple the end points and allow the publishing of services in UDDI directories that can be discovered and invoked dynamically at run time. Unfortunately, at this point in the technology life cycle, the use of WS-Security in the SOAP message header prevents us from being able to do this. Today's Java to WSDL emitters are not yet able to handle the creation of WSDL documents that appropriately describe the WS-Security requirements. Plus, even if they could, at this stage, development tools such as WebSphere Studio Application Developer or Visual Studio .Net couldn't generate the proxies that handle the WS-Security aspects of the service.

As such, the developers of Web services in early 2003 will need to make a conscious trade-off here. When WS-Security is used, the service provider needs to either provide stubs/proxies which partners can invoke that handle the WS-Security portion of the message or manually communicate the WS-Security requirement of the Web service to their potential business partners and customers. For the WS-Security-based project described in this paper, proxies that properly sign the message and insert the WS-Security element into the SOAP data stream were created for Java technology, COM, and .Net clients. The next generation of Web services development tools from IBM and others should be able to handle the WS-Security elements of a Web service, but for now, developers need to understand that this is an achievable, but manual process.

**Summary**

This paper described an Internet-based Web services application that was developed and deployed in 2002. It was deployed on a WebSphere Application Server and is available for use by our customer's business partners. It demonstrates the soundness and overall viability of the draft WS-Security specification by offering itself as a proof-point that secure, mission critical, Web services applications are viable with today's development tools and deployment platforms. Yes, in our customer's case, some non-automated, manual steps were required to handle the WS-Security element of our SOAP message, but as support for WS-Security gets folded into the next iteration of the WSDL specification and support is added to the Web services development tools of many vendors, it will only get better.

# Cross-platform programming with Java technology and the IBM Web Services Toolkit for Mobile Devices

Cross-platform programming with Java technology and the IBM Web Services Toolkit for Mobile Devices

Make sure your Java apps run on as many platforms as possible without modification

**Level:** Introductory
Peter Haggar (haggar@us.ibm.com), Senior Software Engineer, IBM

**18 Feb 2003**

When you're writing a Java application for the server or desktop, you can be reasonably sure that the Java platform will fulfill its "Write Once, Run Anywhere" promise. But when you're dealing with code that will run under the J2ME Mobile Information Device Profile (MIDP), things get a little trickier. If you've downloaded the Web Services Tool Kit for Mobile Devices (WSTKMD) and are eager to write mobile Java Web services, you'll need to check out the warnings in this article(see Resources). Peter Haggar describes some of the pitfalls he encountered in making sure that the WSTKMD's sample applications ran on all the target platforms.

The IBM Web Services Tool Kit for Mobile Devices (see the Resources section below for a link) supports the Java programming language on BlackBerry ™ devices, PocketPC™, and Palm OS4™ mobile devices; on the Palm OS4 platform, it also supports the C programming language. The Java language is touted as a cross-platform environment, with which code can be written once and run on a variety of operating systems without modification. While this is largely true in desktop and server environments, writing Java applications for mobile devices utilizing MIDP (the Mobile Information Device Profile) is not as straightforward.

The sample Java applications that ship with the toolkit run unmodified on all supported devices. However, there the devices have different user interfaces (UIs), and the apps thus have different UIs on the different devices. We took great care to ensure that each Java application would run unmodified on each supported device. This proved more difficult than we first imagined, due to the lack of consistency among the various MIDP implementations, along with differences in the various devices' user interfaces.

In this article, I'll outline the issues that you need to consider, and the programming constructs and style you should use, to ensure that your Java programs can run, unmodified, on each device supported by the IBM Web Services Tool Kit for Mobile Devices.

**MIDP programming**
The MIDP programming environment offers a very limited amount of functionality to the programmer. In addition, MIDP has been implemented differently on different devices. Therefore, you cannot program to MIDP and assume that your application will look and behave consistently in varying environments. For example, the MIDP GUI library does not support a single class for buttons. You can draw buttons natively, or you can rely on each individual MIDP implementation for each device to provide buttons for your menu items.

When creating the sample applications for the Web Services Tool Kit for Mobile Devices, we avoided certain user interface classes in order to ensure that each program would run on each device. When you are preparing your own code for this purpose, you need to accept the fact that your application will have a different interface on different devices. Furthermore, the user of the application will interact with it differently depending on the device on which the program is run.

**Device differences**
The user interacts with each of the devices I'll discuss here differently, due to their differing designs. For the purposes of this article, I'll describe the differences between the devices I worked with during the development of the tool kit: the Palm i705 Handheld running Palm OS4, the Compaq iPAQ 3670 Color Pocket PC running Pocket PC 2002, and the BlackBerry 5810 Handheld running BlackBerry OS 3.2 platform, a Java language operating system. Understanding these differences was key in developing an application that ran on all the devices.

Note that the IBM J9 Virtual Machine that runs on Palm OS4 platform does not support the INETLIB networking library on the Palm i705 device. Therefore, there is no way to make a network call with this VM on this device, and thus the Web Services Tool Kit for Mobile Devices does not support the Palm i705 Handheld. (Other Palm OS4 devices are supported.) However, non-networking applications can still run on the Palm i705 Handheld to help pinpoint user interface issues that arise with Palm OS4 devices.

- **Palm i705 Handheld:** The user interacts with this device via a touchscreen. Menus are also available but are accessed via the menu button provided by the device.
- **iPAQ 3670 Color Pocket PC:** The user interacts with this device via both a touchscreen and a five-way hardware button on the bottom of the device. As you'll see, the touchscreen does not work with all GUI controls available in MIDP, making your choice of GUI control important.
- **BlackBerry 5810 Handheld:** The user interacts with this device via a trackwheel and a hardware selection button. This device does not support a touchscreen and is therefore completely menu driven. Because there is no touchscreen, the process of selecting items is different than it is on the other devices. You use the trackwheel to change the highlighted item, then use the menus or the spacebar to select the highlighted item.

**User interface differences: Menus and buttons**
Some of your design decisions are going to be based on the inherent differences in the devices, as well as the differences in how data is rendered by the different MIDP implementations. For instance, as noted, the BlackBerry device does not have a touchscreen and operates from a menu-driven interface. Therefore, you must design your application to be menu driven if you want it run unmodified on each device.

MIDP contains a Command class that the user can trigger; this class is used to encapsulate operations. Because the BlackBerry device is completely menu driven, all Command objects are displayed as menu choices on its UI. On the other devices, the MIDP implementation determines whether the device displays the Command objects as menu choices or as buttons.

For example, on the Palm i705 Handheld, all Command objects are displayed as menu items. On the iPAQ device, in contrast, they are preferably displayed as buttons; but when there are too many Command objects to fit on the screen, a Menu button is added that, when pressed, displays the remaining Command objects in a menu. In your code, you can determine the order in which these objects appear: when you create Command objects, you create them with a CommandPriority. The lower the number, the higher the priority. On the iPAQ device, the Command objects with the highest priority are given preference to display as buttons rather than menu items.

Because of these differences, when you are designing an application to run unmodified on these various devices, it is important to design the application's flow around the Command class and accept that your Command objects will display either as buttons or menu items depending on the device characteristics.

**User interface: Differences in List implementations**
The differences in GUI controls' appearance on various devices of the sort we just described is to be expected; you can plan around such differences without too much difficulty. However, the differing *behavior* of the same control across devices is cause for more concern.

For example, the List control displays a list of items to be selected. The user is presented with a list of text and/or images. The user can make a selection, and the application takes action based on the item(s) selected. The List control supports three types: Multiple, Exclusive, and Implicit. According to the documentation for MIDP (see Resources), List types work as follows:

- The user can use a List of type Multiple to select one or more items. However, the process of selecting the items does not in itself generate a selection event. The user must take another action, like pressing a button or activating a menu choice, before the program queries the list to determine what item(s) were selected.
- A List of type Exclusive allows the user to select only one item. However, as with the Multiple-type List, the user doesn't generate a selection event by changing the selected item. The programmer must provide a button or menu choice to enable the user to confirm the choice made; activating this choice or button prompts the program to query the List and determine which item was selected.
- The Implicit-type List is the only List that is supposed to automatically generate a selection event when an item in the List is selected. Only one item may be selected at a time, and each time the selection item is changed, a selection event is automatically generated.

Unfortunately, as implemented on the three platforms I'm describing here, the List control does not always demonstrate the behavior documented for it. Furthermore, the behavior of the List control differed from platform to platform, making it impossible to use this control in any cross-platform applications. Let's look at these inconsistencies in detail.

**List control on Palm OS4 devices**
Under Palm OS4 platform, each time you select an item in a Multiple- or Exclusive-type List, a selection event is generated. This behavior is not consistent with the MIDP documentation.
For an Implicit-type List, at no time are selection events generated. To use this type of List under Palm OS4 platform, you would need to provide a menu item that, when selected by the user, prompts the app to query the list to see which item has been selected. This behavior is also not consistent with the MIDP documentation; a programmer writing an application to the MIDP spec would probably not provide such a menu item, thus making the Implicit useless on this platform.

**List control on Compaq iPAQ Pocket PC with PocketPC 2002**
Under Pocket PC 2002 platform on the Compaq iPAQ device, for both Multiple- and Exclusive-type Lists, no events are generated when you select List items. This behavior is consistent with the MIDP documentation.

For an Implicit-type List, however, there are problems. If you attempt to select an item using the stylus, the iPAQ device's touchscreen will not properly change the item selection. If you use the five-way hardware button, you can select items properly; pressing the center of the five-way hardware button then generates a selection event. This behavior is consistent with the MIDP documentation for List; however, the lack of touchscreen control with this type of List is bothersome, and is most likely a bug in the MIDP implementation for this device.

**List control on BlackBerry 5810 Handheld**
Because the BlackBerry device does not have a touchscreen, items are highlighted by moving the

trackwheel. The user selects the highlighted item by pressing the spacebar or using menu options. For a Multiple-type List on the BlackBerry 5810 Handheld, no selection events are generated when pressing the spacebar to select an item. In addition, choosing Change Option from the menu has the same effect as pressing the spacebar: It changes the selected item but does not generate an event. This behavior is consistent with the MIDP documentation.

For an Exclusive-type List, no selection events are generated when pressing the spacebar to select an item. However, in contrast with the Multiple-type List, there were no menu items to choose from. This behavior is consistent with the MIDP documentation, though inconsistent with the implementation of the Multiple-type List.

For an Implicit-type List, selection events are generated when you choose Select from the menu. Pressing the spacebar has no effect. This behavior is consistent with the MIDP documentation.
Differing List controls: Can they be reconciled?

The List control behavior differs from device to device. In addition, its behavior on the Palm device is not consistent with its documented behavior. On the iPAQ device, the List control's behavior is consistent with the MIDP documentation; however, the fact that the Implicit-type List does not respond to the touchscreen and requires the use of the five-way hardware button is inconsistent and aggravating. The BlackBerry implementation is also consistent with the documentation; however, the menus available under the three types of List are not consistent. The Multiple type contains a Change Option menu item, the Exclusive type contains no menu items, and the Implicit type contains a Select menu item.

Other MIDP implementations may also provide inconsistent behavior; you should not assume that they all have the same limitations described here.

**ChoiceGroup: An alternative to List**
Because of the problems associated with the List control on the devices I used, I realized that I could not use that control in a cross-platform application. Instead, I use the ChoiceGroup control to display a list of choices to the user. You can create either a Multiple- or an Exclusive-type ChoiceGroup control. Like their corresponding List types, these types should not generate selection events when the user chooses items. For the sample applications, I used an Exclusive-type ChoiceGroup, since I only wanted the user to be able to make one selection from each list. Then, I provided different Command objects that, when selected, would take action based on which item in the ChoiceGroup was selected.

**Multithreaded connections**
There is one more trick you'll need to keep in mind in order to have one program execute on all three devices we're considering here. A device running the BlackBerry OS can only make an HTTP connection from a secondary thread and not on the main thread. Palm OS4 and Pocket PC 2002 devices have no such restriction.

I initially attempted to have my applications dynamically determine which device they were running on and create a secondary thread only when running on the BlackBerry platform. However, this technique itself ran into a problem: the System.getProperty("os.version"); call returned a null value on the BlackBerry platform. Under the Pocket PC 2002 and Palm OS4 operating systems, this method call returned valid values; however, I did not want to assume that a null return from a System.getProperty("os.version"); call always implied the need for a secondary thread for HTTP connections.

Therefore, the code for the Palm and Pocket PC device becomes slightly more complicated with the addition of a secondary thread used only to make the HTTP connection. The code in the sample programs shows how this is done. However, if you don't plan to deploy your application to the BlackBerry platform, you won't need to worry about this issue.

**Conclusion**

Writing one Java application that will run intuitively for the user on Palm OS4, Pocket PC 2002, and BlackBerry OS 3.2 platforms is not as straightforward as desktop or server Java programming. The MIDP environment is not as robust as J2SE or J2EE, and the differences among devices can play havoc with your cross-platform plans. You need to take care to ensure that the GUI controls you use work properly and somewhat consistently on each deployed platform. In addition, the user interface requirements must not exceed the various devices' capabilities.

To some extent, you must take a least-common-denominator approach and work around the various characteristics and limitations of your target devices to create an application that is usable on each of them. In addition, you might have to add complexities that are unnecessary on some platforms in order to satisfy the requirements of others. For example, the addition of a secondary thread for HTTP connections for BlackBerry OS platforms makes code more complex than necessary for Palm OS4 and Pocket PC 2002 devices.

In the end, it is worth putting in the additional work and complexity to have one application run on multiple devices. Doing so reduces code maintenance and the time and effort needed to track multiple versions of the same application. Furthermore, this approach takes advantage of the cross-platform features of the Java programming language. Over time, as J2ME and MIDP improve, the differences between the various MIDP implementations and their behavior on the various devices will diminish. This will result in an easier and more straightforward path to cross-platform J2ME and MIDP development.

# Understanding quality of service for Web services
# Improving the performance of your Web services

**Level: Introductory**

Anbazhagan Mani (manbazha@in.ibm.com), Software engineer, IBM Software Labs, India
Arun Nagarajan (anagaraj@in.ibm.com), Software engineer, IBM Global Services India

**01 Jan 2002**

With the widespread proliferation of Web services, quality of service (QoS) will become a significant factor in distinguishing the success of service providers. QoS determines the service usability and utility, both of which influence the popularity of the service. In this article, we look at the various Web service QoS requirements, bottlenecks affecting performance of Web services, approaches of providing service quality, transactional services, and a simple method of measuring response time of your Web services using the service proxy.

The dynamic e-business vision calls for a seamless integration of business processes, applications, and Web services over the Internet. Delivering QoS on the Internet is a critical and significant challenge because of its dynamic and unpredictable nature. Applications with very different characteristics and requirements compete for scarce network resources. Changes in traffic patterns, denial-of-service attacks and the effects of infrastructure failures, low performance of Web protocols, and security issues over the Web create a need for Internet QoS standards. Often, unresolved QoS issues cause critical transactional applications to suffer from unacceptable levels of performance degradation.

With standards like SOAP, UDDI, and WSDL being adopted by all major Web service players, a whole range of Web services — covering the financial services, high-tech, and media and entertainment — are being currently developed. As most of the Web services are going to need to establish and adhere to standards, QoS will become an important selling and differentiating point of these services.

QoS covers a whole range of techniques that match the needs of service requestors with those of the service provider's based on the network resources available. By QoS, we refer to non-functional properties of Web services such as performance, reliability, availability, and security.

## Web service QoS requirements

The major requirements for supporting QoS in Web services are as follows:

- **Availability**: Availability is the quality aspect of whether the Web service is present or ready for immediate use. Availability represents the probability that a service is available. Larger values represent that the service is always ready to use while smaller values indicate unpredictability of whether the service will be available at a particular time. Also associated with availability is time-to-repair (TTR). *TTR* represents the time it takes to repair a service that has failed. Ideally smaller values of TTR are desirable.
- **Accessibility**: Accessibility is the quality aspect of a service that represents the degree it is capable of serving a Web service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There could be situations when a Web service is available but not accessible. High accessibility of Web services can be achieved by building highly scalable systems. *Scalability* refers to the ability to consistently serve the requests despite variations in the volume of requests.
- **Integrity**: Integrity is the quality aspect of how the Web service maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions will provide the correctness of interaction. A *transaction* refers to a sequence of activities

to be treated as a single unit of work. All the activities have to be completed to make the transaction successful. When a transaction does not complete, all the changes made are rolled back.

- **Performance**: Performance is the quality aspect of Web service, which is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web service. *Throughput* represents the number of Web service requests served at a given time period. *Latency* is the round-trip time between sending a request and receiving the response.
- **Reliability**: Reliability is the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality. The number of failures per month or year represents a measure of reliability of a Web service. In another sense, reliability refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers.
- **Regulatory**: Regulatory is the quality aspect of the Web service in conformance with the rules, the law, compliance with standards, and the established service level agreement. Web services use a lot of standards such as SOAP, UDDI, and WSDL. Strict adherence to correct versions of standards (for example, SOAP version 1.2) by service providers is necessary for proper invocation of Web services by service requestors.
  **Security**: Security is the quality aspect of the Web service of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control. Security has added importance because Web service invocation occurs over the public Internet. The service provider can have different approaches and levels of providing security depending on the service requestor.

**QoS enabled Web services**

The interface definition (WSDL) specifies the syntactic signature for a service but does not specify any semantics or non-functional aspects. QoS enabled Web services require a separate QoS language for Web services to answer the following questions:

- What's the expected latency?
- What's the acceptable round-trip time?

A programmer needs to be able to understand the QoS characteristics of the Web services while developing applications that invoke Web services.

Ideally, a QoS enabled Web services platform should be capable of supporting a multitude of different types of applications :

1. With different QoS requirements
2. By making use of different types of communication and computing resources

When considering QoS-aware Web services, we suppose that the interface specifications are extended with statements on QoS that can be associated to the whole interface or to individual operations and attributes. In the case of a service requestor, these statements describe the required QoS associated with the service required by the client, while from a service provider's perspective these statements describe the offered QoS associated with the service offered by the server object.

The Web service architecture design from IBM includes a separate layer called "endpoint description" to add additional semantics to service description like QoS properties.

### QoS negotiation & binding establishment

The following steps should be performed during binding establishment using a QoS-enabled Web services platform:

1. The service requestor requests the establishment of the binding by specifying the reference to a Web service interface. This request also contains the required QoS.
2. The QoS broker searches for the service providers in the UDDI.
3. The QoS broker performs QoS negotiation as described below.
4. The Web service QoS broker compares the offered QoS with the required QoS and uses its internal information to determine an agreed QoS. This process is called QoS negotiation.
5. If the QoS negotiation has been successful, the service requestor and service provider are informed that a negotiation has been successful and a binding has been built. From this moment on these objects can interact through the binding.

### Bottlenecks in performance of Web services

Web services can encounter performance bottlenecks due to the limitations of the underlying messaging and transport protocols. The reliance on common widely accepted protocols such as HTTP and SOAP, however, make them a permanent burden that must be shouldered. Thus it is important to understand the workings of these limitations.

### HTTP

HTTP is a best-effort delivery service. It is a stateless data-forwarding mechanism which tends to create two major problems:

- There is no guarantee of packets being delivered to the destination.
- There is no guarantee of the order of the arriving packets.

If there is no bandwidth available, the packets are simply discarded. Bandwidth is clearly a bottleneck, as users and amounts of data running over the network increase. Traditionally, many applications assume zero latency and infinite bandwidth. Also traditionally, applications use synchronous messaging. Synchronous messaging is fine when you run an application on your own computers; components communicate with latencies measured in microseconds. However, with Web services, they communicate across the Internet, which means latencies are measured in tens, hundreds, or even thousands of milliseconds.

Although newly designed protocols like Reliable HTTP (HTTPR), Blocks Extensible Exchange Protocol (BEEP), and Direct Internet Message Encapsulation (DIME) can be used, widespread adoption of these new protocols for Web service transport like HTTPR and BEEP will take some time. Hence, application designers making use of Web services should understand performance issues of Web service such as latency, and availability while designing their systems. Some of the ways to improve Web service performance are given below.

### Use of asynchronous message queues

Applications which rely on remote Web services can use message queuing to improve reliability, but at the cost of response time. Applications and Web services within an enterprise can use message queuing like Java Messaging Service (JMS) or IBM MQSeries for Web Service invocations. Enterprise messaging provides a reliable, flexible service for the asynchronous exchange of critical data

throughout an enterprise. Message queues provide two major advantages:

1. It is asynchronous: A messaging service provider can deliver messages to the requestor as they arrive and the requestor does not have to request messages in order to receive them.
2. It is reliable: A messaging service can ensure that a message is delivered once and only once.

In the future, *Publish & Subscribe* messaging systems over the Internet such as the Utility Services package from alphaWorks , can be used for Web service invocations (see Resources).

**Private WANs and Web service networks**
Use of private WANs/extranets and Web services networks can be a suitable option for businesses depending on Web services which are mission-critical. These private networks provide low network latency, low congestion, guaranteed delivery, and non-repudiation. However, in some cases it could be costly to have a private network.

**SOAP and performance**
SOAP is the defacto wire protocol for Web services. SOAP performance is degraded because of the following:

- Extracting the SOAP envelope from the SOAP packet is time-expensive.
- Parsing the contained XML information in the SOAP envelope using a XML parser is also time-expensive.
- There is not much optimization possible with XML data.
- SOAP encoding rules make it mandatory to include typing information in all the SOAP messages sent and received.
- Encoding binary data in a form acceptable to XML results in overhead of additional bytes added as a result of the encoding as well as processor overhead performing the encoding/decoding.

The XML processor must be loaded, instantiated, and fed with the XML data. Then the method call argument information must be discovered. This involves a lot of overhead as XML processors grow to support more XML features.

**The role of the XML parser in SOAP performance**

Most existing XML parsers are too expensive in terms of code size, processing time, and memory foot print because these parsers have to support a number of features like type checking and conversion, wellformedness checking, or ambiguity resolution. All these make XML parsers require more computing resources. Some applications can consider using of stripped down version of XML parser which have a small code size and memory foot print.

Also, most of the current SOAP implementations are Document Object Model (DOM) based. DOM parsers are inherently slow to parse the messages. SAX-based SOAP implementations can be used to increase throughput, reduce memory overhead, and improve scalability.

**Compressing XML**
SOAP uses XML as its payload. And if we consider thousands of SOAP messages being transmitted over the Web, the network bandwidth is being pushed to its limit. XML's way of representing data usually results in a substantially larger size than representing the same data in binary, which is on average 400% larger. This increase of the message size creates a critical problem when data has to be transmitted quickly, which effectively results in increase of the data transmission time. Some application designs should consider techniques for compact and efficient representation. One of the

ways to achieve this can be to compress the XML — especially when the CPU overhead required for compression is less than the network latency.

**Other factors affecting Web service performance**
There are still more factors that can affect Web service performance that are outside the control of the Web service application, such as:

- Web server response time and availability.
- Original application execution time like EJB/Servlets in Web application server.
- Back-end database or legacy system performance.

**Approaches to provide proactive Web service QoS**
Service providers can proactively provide high QoS to the service requestors, by using different familiar approaches like caching and load balancing of service requests. Caching and load balancing can be done at both Web server level and at Web application server level. Load balancing prioritize various types of traffic and ensure that each request is treated appropriately to the business value it represents.

A Web service provider can perform capacity modeling to create a top-down model of request-traffic, current capacity utilization, and the resulting QoS. A service provider can also categorize Web service traffic by the volume of traffic, traffic for different application service categories, and traffic from different sources. This will help in understanding the capacity that will be required to provide good QoS for a volume of service demand and for future planning like capacity and type of load balancing Web application servers and/or Web servers (for example, the number of servers required for setting up a clustered server farm).

Service providers can provide *differentiated servicing* by using the capacity model to determine the capacity needed for different customers and service types and by ensuring appropriate QoS levels for different applications and customers. For example, a multimedia Web service might require good throughput, but a banking Web service might require security and transactional QoS.

**Transactional QoS**
Transactional QoS refers to the level of reliability and consistency at which the transactions are executed. Transactional QoS is crucial for maintaining the integrity of a Web service. Transactions are very important for business processes to guarantee that a set of related activities are treated and completed as a single unit of work. If an exception occurs while executing a transaction, the transaction has to be recovered back to a consistent state. This property is called the "atomicity" of a transaction. Other than property of atomicity, transactions in a stricter sense should satisfy consistency, isolation and durability properties. All these four properties are together called "ACID" properties (see the sidebar).

There are several approaches to provide transactional QoS. The most popular approach, which is traditionally used in Web application architectures is the two-phase commit. Two-phase commit provides a transaction coordinator which controls the transaction based on the idea that no constituent transaction is allowed to commit unless they are all able to commit. This approach of using a transaction coordinator to ensure atomicity is used in Java Transactional Service (JTS), CORBA OTS, and in most database management systems.

But there are new complications when we are thinking of transactions involving Web services. The Web services used by a particular application or Web service are often distributed remotely over the Web as well as owned by different parties. Having a central transaction coordinator, which dictates the rules and performs the commits and rollbacks, in a Web services environment is very tedious to

implement considering the transaction coordinator does not have full control over all the resources. Also, two-phase commit protocol involves one or other form of resource locking. Longer periods of resource locking will result in serious scalability issues. Therefore even though it is possible to use, extreme care should be taken to make sure that resources are not locked for long periods of time. The OASIS Business Transactions technical committee has released the Business Transaction Protocol (BTP) which extends the two-phase commit transaction management approach to address the needs of disparate trading partners that use XML to exchange data over the Web. BTP allows both the normal ACID transactions and non-ACID transactions (termed as "cohesions") for long lasting transactions that span multiple enterprises.

Another approach called compensation is based on the idea that a transaction is always allowed to commit, but its effect and actions can be cancelled after it has committed. For example, it may be possible to order a shipment, and then later cancel the shipment if the required shipment process has not yet started. Canceling the shipment is an example of a compensating transaction; it compensates for the initial transaction that ordered the shipment. In compensating transaction, each "real" transaction has an associated "compensating" transaction. This "compensating" transaction element describes a way to revert changes done by the "real" transaction and return to a previous consistent state. If any transaction aborts, the caller executes the corresponding compensating transaction for all the transactions that have previously committed. Two major problems associated with compensating transactions are:

- Compensating transactions, unlike two-phase commit, may not satisfy all the four "ACID" properties at all times — this means there is always a probability for a failure.
- Traditionally designed two-phase commit transactions have to be redesigned to provide way for compensation.

**A simple method to measure response time of your Web services**

A simple method to measure the performance characteristics of your Web services can be developed by adding a little bit of extra functionality in the service proxy. Service proxies in Web services are similar to stubs in Java RMI. They contain the code that is specific to a binding within the service interface, thereby hiding the complex network communications details from the client. For example, if the binding is a SOAP binding, then the service proxy will contain SOAP-specific code that can be used by the client to invoke the service.

The steps involved in developing a proxy capable of measuring response time is as follows:

1. Generate service proxy from the WSDL service definition file.
2. Modify the generated service proxy to add code to clock the time (see Listing 2).
3. Re-compile the modified service proxy.
4. Develop a client program to create a object of the service proxy and invoke the necessary methods .

**Step 1: Generate a service proxy from service definition**

Typically, service proxies are not written by the programmer. Service proxies can be easily generated from the WSDL file. Web Service Toolkits (including the alphaWorks WSTK) provide tools to generate service proxies (see the sidebar). A sample WSDL service definition for an EchoService is given in Listing 1. This is a simple Web service, which echos back the original string with "Hello" appended to it.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="EchoService"
 targetNamespace="http://www.echoserviceservice.com/EchoService-interface"
```

```
 xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.echoserviceservice.com/EchoService"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<message name="InechoRequest">
 <part name="meth1_inType1" type="xsd:string"/>
</message>
<message name="OutechoResponse">
 <part name="meth1_outType" type="xsd:string"/>
</message>
<portType name="EchoService">
 <operation name="echo">
   <input message="InechoRequest"/>
    <output message="OutechoResponse"/>
 </operation>
</portType>
<binding name="EchoServiceBinding" type="EchoService">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="echo">
   <soap:operation soapAction="urn:echoservice-service"/>
   <input>
     <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="urn:echoservice-service"
        use="encoded"/>
   </input>
    <output>
     <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="urn:echoservice-service" use="encoded"/>
   </output>
 </operation>
</binding>
<service
    name="EchoService">
 <documentation>IBM WSTK 2.0 generated service definition file</documentation>
 <port binding="EchoServiceBinding" name="EchoServicePort">
   <soap:address location="http://localhost:8080/soap/servlet/rpcrouter"/>
 </port>
</service>
</definitions>
```

**Step 2: Modify the generated service proxy**

Even though the machine-generated Service Proxy code is not to be edited, let us slightly bend this rule by adding a few lines of code. These added lines instantiates a Timer object to measure the time it takes to bind to the server and invoke a method. This is illustrated in the sample code given in listing2.

```
import java.net.*;
import java.util.*;
import org.apache.soap.*;
```

```
import org.apache.soap.encoding.*;
import org.apache.soap.rpc.*;
import org.apache.soap.util.xml.*;
import mytimer.Timer;
public class EchoServiceProxy
{
 private Call call = new Call();
 private URL url = null;
 private String SOAPActionURI = "";
 private SOAPMappingRegistry smr = call.getSOAPMappingRegistry();
 public EchoServiceProxy() throws MalformedURLException
 {
  call.setTargetObjectURI("urn:echoservice-service");
   call.setEncodingStyleURI("http://schemas.xmlsoap.org/soap/encoding/");
   this.url = new URL("http://localhost:8080/soap/servlet/rpcrouter");
   this.SOAPActionURI = "urn:echoservice-service";
 }
 public synchronized void setEndPoint(URL url)
 {
   this.url = url;
 }
 public synchronized URL getEndPoint()
 {
   return url;
 }
 public synchronized java.lang.String echo
   (java.lang.String meth1_inType1) throws SOAPException
 {
  if (url == null)
   {
     throw new SOAPException(Constants.FAULT_CODE_CLIENT,
    "A URL must be specified via " +
    "EchoServiceProxy.setEndPoint(URL).");
   }
  call.setMethodName("echo");
   Vector params = new Vector();
  Parameter meth1_inType1Param = new Parameter("meth1_inType1",
    java.lang.String.class, meth1_inType1, null);
  params.addElement(meth1_inType1Param);
  call.setParams(params);

   // Start a Timer
  Timer timer = new Timer();
   timer.start();

   Response resp = call.invoke(url, SOAPActionURI);

   // Stop the Timer
   timer.stop();
  // Print the response time by calculating the difference
  System.out.println("Response Time = " + timer.getDifference());
```

```
    // Check the response.
    if (resp.generatedFault())
    {
      Fault fault = resp.getFault();
      throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
   }
    else
{
      Parameter retValue = resp.getReturnValue();
     return (java.lang.String)retValue.getValue();
    }
  }
}
```

**Step 3: Re-compile the modified service proxy**
The modified service proxy source file has to be recompiled simply by using javac command or by using any other compiler.

**Step 4: Develop a client program**
Develop a client application, which can use the service proxy to invoke the Web service. This could be a simple Java program or a AWT/Swing based Java GUI application.

**Conclusion**
Quality of services is an important requirement of business-to-business transactions and thus a necessary element in Web services. The various QoS properties such as availability, accessibility, integrity, performance, reliability, regulatory, and security, need to be addressed in the implementation of Web service applications. The properties become even more complex when you add the need for transactional features to Web services. Some of the limitations of protocols such as HTTP and SOAP may hinder QoS implementation, but there are a number of ways to provide proactive QoS in Web services.

# Advertise with JavaJazzUp

We are the top most providers of technology stuffs to the java community. Our technology portal network is providing standard tutorials, articles, news and reviews on the Java technologies to the industrial technocrats. Our network is getting around 3 million hits per month and its increasing with a great pace.

For a long time we have endeavored to provide quality information to our readers. Furthermore, we have succeeded in the dissemination of the information on technical and scientific facets of IT community providing an added value and returns to the readers.

We have serious folks that depend on our site for real solutions to development problems.

**JavaJazzUp Network** comprises of :

http://www.roseindia.net
http://www.newstrackindia.com
http://www.javajazzup.com
http://www.allcooljobs.com

**Advertisement Options:**

| Banner | Size | Page Views | Monthly |
|---|---|---|---|
| Top Banner | 470*80 | 5,00,000 | USD 2,000 |
| Box Banner | 125 * 125 | 5,00,000 | USD 800 |
| Banner | 460x60 | 5,00,000 | USD 1,200 |
| Pay Links | | Un Limited | USD 1,000 |
| Pop Up Banners | | Un Limited | USD 4,000 |

The http://www.roseindia.net network is the "real deal" for technical Java professionals. Contact me today to discuss your customized sponsorship program. You may also ask about advertising on other Technology Network.

Deepak Kumar
deepak@roseindia.net

# Valued JavaJazzup Readers Community

**We invite you to post Java-technology oriented stuff. It would be our pleasure to give space to your posts in JavaJazzup**.

**Contribute to Readers Forum**

If theres something youre curious about, were confident that your curiosity, combined with the knowledge of other participants, will be enough to generate a useful and exciting Readers Forum. If theres a topic you feel needs to be discussed at JavaJazzup, its up to you to get it discussed.

**Convene a discussion on a specific subject**

If you have a topic youd like to talk about . Whether its something you think lots of people will be interested in, or a narrow topic only a few people may care about, your article will attract people interested in talking about it at the Readers Forum. If you like, you can prepare a really a good article to explain what youre interested to tell java technocrates about.

**Sharing Expertise on Java Technologies**

If youre a great expert on a subject in java, the years you spent developing that expertise and want to share it with others. If theres something youre an expert on that you think other technocrates might like to know about, wed love to set you up in the Readers Forum and let people ask you questions.

**Show your innovation**

We invite people to demonstrate innovative ideas and projects. These can be online or technology-related innovations that would bring you a great appreciations and recognition among the java technocrates around the globe.

**Hands-on technology demonstrations**

Some people are Internet experts. Some are barely familiar with the web. If you'd like to show others aroud some familiar sites and tools, that would be great. It would be our pleasure to give you a chance to provide your demonstrations on such issues : How to set up a blog, how to get your images onto Flickr, How to get your videos onto YouTube, demonstrations of P2P software, a tour of MySpace, a tour of Second Life (or let us know if there are other tools or technologies you think people should know about...).

**Present a question, problem, or puzzle**

Were inviting people from lots of different worlds. We do not expect everybody at Readers Forum to be an expert in some areas. Your expertise is a real resource you may contribute to the Java Jazzup. We want your curiosity to be a resource, too. You can also present a question, problem, or puzzle that revolves around java technologies along with their solution that you think would get really appreciated by the java readers around the globe.

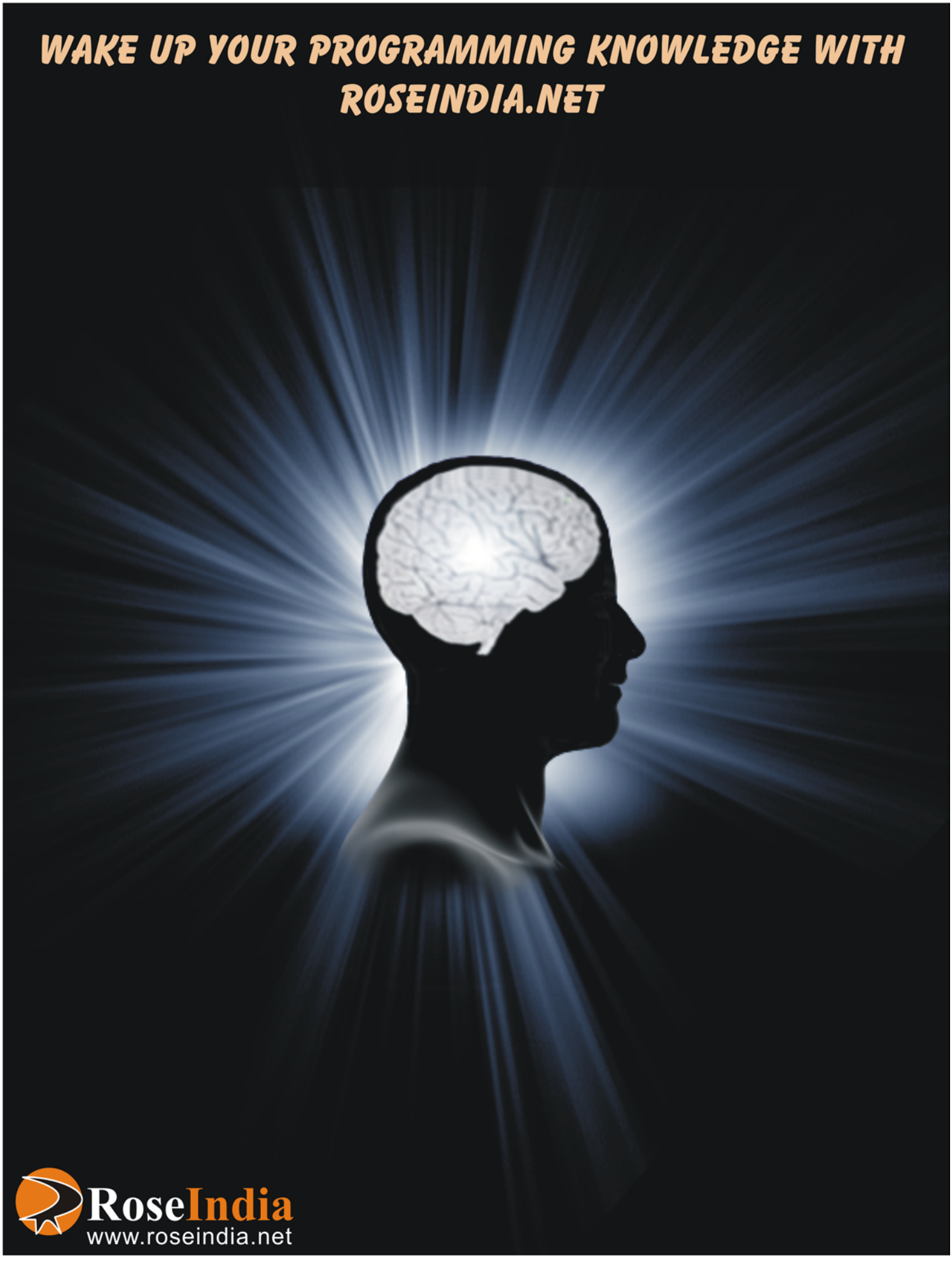
**Post resourceful URLs**

If you think you know such URL links which can really help the readers to explore their java skills. Even you can post general URLs that you think would be really appreciated by the readers community.

**Anything else**

If you have another idea for something youd like to do, talk to us. If you want to do something that we havent thought of, have a crazy idea, wed really love to hear about it. Were open to all sorts of suggestions, especially if they promote readers participation.

# WAKE UP YOUR PROGRAMMING KNOWLEDGE WITH ROSEINDIA.NET



RoseIndia
www.roseindia.net

NEWSTRACK india

**Broadcasters are storytellers, newspapers are fact providers and we serve the both.**

http://www.newstrackindia.com