Mock Web services with Apache Synapse to develop and test Web services

Do more with Apache Synapse Enterprise Service Bus

Skill Level: Intermediate

Upul Godage (upulgodage@yahoo.com)
Senior Software Engineer
WSO2

01 May 2008

Apache Synapse is a simple, lightweight, high-performance enterprise service bus (ESB) released under the Apache License, Version 2.0 from the Apache Software Foundation. Using Apache Synapse, you can filter, transform, route, manipulate, and monitor SOAP, binary, XML, and plain text messages that pass through your large-scale enterprise systems by HTTP, HTTPS, Java™ Message Service (JMS), Simple Mail Transfer Protocol (SMTP), Post Office Protocol Version 3 (POP3), FTP, file systems, and many other transport mediums. But for an individual developer, what’s the use of an ESB product in your day-to-day life? The simplicity of the configuration, out-of-the-box feature set, extensible architecture, and the minimal footprint makes it a versatile and powerful tool that you can use for a variety of tasks. This article examines how you can use Apache Synapse to create mock Web services.

Section 1. Before you start

About this tutorial

This tutorial discusses how to create mock Web services using Apache Synapse ESB. The target Web service clients and services can be in any language, such as Microsoft® .NET, Java, or PHP. You work with several samples, starting from the
most basic configuration and gradually building more complex solutions to create mock Web services.

Prerequisites

This tutorial is written mainly for Web services developers, so you should have a general familiarity with Web services concepts. Prior knowledge of Apache Synapse is helpful. You should also have at least an elementary knowledge of XSL Transformation (XSLT) and JavaScript code. If this is an unfamiliar area to you, there are many resources on the Web that can provide you with the basic information.

System requirements

Apache Synapse requires JDK 1.5 and can run in Linux®, Microsoft Windows®, and Solaris environments. The memory and disk space requirements are minimal, as you can easily run it on low-end systems without any issues.

In this tutorial, commands and directory names are given in a Linux-compatible format. If your operating system is different, please change the commands and the directory names accordingly. For example, in Windows you have to use the synapse.bat script instead of synapse.sh to start the Synapse instance.

Section 2. What are mock Web services, and why should I use them?

You might encounter the following four scenarios in your daily work:

- You’re asked to write a program that depends on one or more remote Web services. The Web services can be running in production servers, and you might not have them readily available in test servers. For the third-party services, you might not even have access to the code or the binaries. So you might have to assemble some fake services from the ground up to test your code. Chances are that you’re not testing the bug fixes you’ve made to your hotel reservation system against the production servers by putting reservations and canceling them afterwards. Or are you?
- You might be working at home, on the road, or while flying, and you need
a full set of working Web services offline so you can try out your latest changes to the order-processing system. This system checks products through a catalog service and places the order through an order-processing service.

- You want to create an offline demo, but you're not sure that the remote Web services will be accessible and running all the time. You need to make sure that the Web service calls respond properly—that is, no connection timeout exceptions—when potential customers are sitting in front of you.

- You might need to simulate different scenarios. What if the service responds with a fault or some unexpected response? It would be hard to get the exact responses you want from hosted test services. Your program might not have been tested with a rooms not available fault response until it happens after two weeks in production.

In all these scenarios you can solve these problems by creating fake Web services, called mock Web services, that mimic the behaviour of real Web services. Mock Web services have the same interface and behaviour as the real Web services. At development and testing times, you swap the real Web service endpoints with the mock Web service endpoints. This lets you validate the request messages and feed in the required response messages for the external Web service calls.

Section 3. The Apache Synapse solution

The great thing about using Apache Synapse for mock Web services is that you can apply the same skills and know-how you've gained using the ESB when solving enterprise integration problems to this task without learning and investing in another tool. You can transform and manipulate the responses the way you want using available tools, scripting languages, stylesheets, and even Java code.

I presume that you've installed and have run some samples of Apache Synapse. Apache Synapse comes with plenty of samples that you can try out with the help of the bundled sample server and client (see the Downloads section). If you haven't done so, go to the Apache Synapse site and download the latest release (see the Resources section for the link). Check out the samples guide, and try some of the samples. By following the step-by-step instructions in the guide you can get started in no time. In this article, you're going to use the same sample client and simulate the same sample server behavior in Apache Synapse using some Synapse configurations.

The Synapse configuration is an XML file written in a domain-specific language that
you use to tell Synapse what to do with the received messages. (See the Synapse configuration language guide in the Resources section for the complete syntax of the Synapse configuration.)

You’re going to use simple XSL stylesheets and scripting code in these examples. Even if you’re not familiar with them, you can modify the given samples and use them to suit your requirements easily.

Section 4. Set up the samples

First you install Apache Synapse and prepare the environment to try out the samples:

1. Download the latest release of Apache Synapse, and extract the distribution to a directory (see Resources for a link to the Apache Synapse download).
2. If you don’t have it already, install JDK 1.5 and point the JAVA_HOME environment variable to it.
3. Install Apache Ant and the Apache Ant bin directory, available in the PATH environment variable (see Resources for the link).
4. Download the sample package from the Downloads section, and extract it to a directory.
5. From the samples package, copy the mocks directory to the Apache Synapse repository/conf/sample/resources directory. Figure 1 shows where to copy the files in the Synapse directory structure.
6. Copy the five Synapse configuration files to the Apache Synapse repository/conf/sample directory (see Figure 1).

Figure 1. Location of sample files
You use the same mechanism that Synapse is using to start the bundled sample configurations to try out the Synapse sample configuration files. That’s why the configurations are named synapse_sample_*.xml and located in the sample directory.

The sample client bundled with the Synapse distribution has been written to
generate different types of requests, process their responses, and display the output. (For more details, see the Synapse samples guide in the Resources section.)

You can use a tool like Apache TCPMon to see what happens when you run the client (see Resources for the link). Configure the monitoring tool to listen on 8081, forwarding to port 8080. You're setting up the tool to intercept the messages moving between the client and Synapse. TCPMon shows what the client is sending and what Synapse is sending back. If you're not using TCPMon, replace port 8081 with 8080 when you're running the sample client commands later.

That's it. Now you can start building mock Web services with Apache Synapse. You start with the simplest sample and build more complex configurations gradually.

Section 5. Simple request and response

First you simulate the simplest scenario: Instruct Apache Synapse to send a fixed response no matter what request it receives:

1. Start Synapse with the synapse_sample_mock1-simple.xml configuration by going to the Synapse bin directory and giving the command

```
./synapse.sh -sample mock1-simple
```

This starts up the Synapse instance with the synapse_sample_mock1-simple.xml configuration, which you copied earlier to repository/conf/sample.

2. After Synapse starts, go to the Synapse samples/axis2Client directory, and run the sample client like this:

```
ant stockquote
-Dtrpurl=http://localhost:8081/
```

The sample client creates a getQuote operation request for symbol IBM and sends it to the endpoint address, http://localhost:8081/soap/StockQuote. Then it prints the stock price extracted from the response received. If you've used a tool like TCPMon, you can see that the client is sending a getQuote operation request message and that Synapse is sending back a getQuoteResponse response message.

Note that this is similar to the sample 0 in the Synapse samples guide (see Resources for a link). But you didn't use the Addressing headers. That's to keep the sample as simple as possible. You haven't used the sample server; instead, Synapse has simulated the server response. Listing 1 shows the Synapse configuration you've used.

Listing 1. Synapse configuration for simple request and response:
In the configuration, you’ve replaced the request message contents with the response message contents using a simple stylesheet transformation and marked it as a response so that Synapse delivers it back to the requester. You also removed the WS Addressing header if it exists. Otherwise, Synapse uses that header to deliver the message. In this simplest case, whatever the request is, you’ve configured Synapse to send this fixed message back. You can try calling this endpoint with different request messages; it will reply with the same response message. You can change the message contents in the stylesheet and change the Synapse response. Now run the sample client as follows using ant stockquote
-Dtrpurl=http://localhost:8081/ -Dmode=fullquote -Dsymbol=ABC. This sends a full stock quote request for the symbol ABC.

If you monitor the request message, you can see that this time it’s a getFullQuote operation request message with the stock symbol ABC. But Synapse is sending the same response as earlier.

Listing 2 is the stylesheet, GetQuoteResponse.xsl, you used. You’ve simply replaced the current request message with the response message contents.

Listing 2. Stylesheet for simple request and response: GetQuoteResponse.xsl

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <soapenv:Envelope
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
        <ns:getQuoteResponse
          xmlns:ns="http://services.samples/xsd">
          <ns:return
            type="samples.services.GetQuoteResponse">
            <ns:change>-2.920874334221886</ns:change>
            ...
            <ns:symbol>IBM</ns:symbol>
            <ns:volume>9343</ns:volume>
          </ns:return>
        </ns:getQuoteResponse>
      </soapenv:Body>
    </soapenv:Envelope>
  </xsl:template>
</xsl:stylesheet>
```
In Listing 2, you've defined an implicit main sequence. All mediator elements that reside in the top level, such as the in mediator in this instance, belong to that. Every request that Synapse receives has gone through this sequence. In the following sample you host a virtual Web service, which has its own endpoint address and its own Web Services Description Language (WSDL).

Section 6. Set up a proxy service

This time you host a virtual Web service in Synapse that returns a predefined response message like the previous sample. Also, the virtual Web service publishes your, custom WSDL simulating the real Web service. This sample is similar to the simple proxy service, sample 150, in the Synapse samples guide (see Resources for the link).

1. Start Synapse with the synapse_sample_mock2-proxy.xml configuration like this: ./synapse.sh -sample mock2-proxy.

2. To view the Web service WSDL, go to http://localhost:8080/soap/StockQuoteService?wsdl in a Web browser. This is the custom WSDL you've given for the proxy service.


4. You get the stock price as in the previous sample. In this case, you've sent the request message to the location where the proxy service is hosted. Listing 3 is the Synapse configuration you used.

Listing 3. Synapse configuration for setting up a proxy service: synapse_sample_mock2-proxy.xml

```xml
<definitions xmlns="http://ws.apache.org/ns/synapse">
    <localEntry key="GetQuoteResponse_xsl" src="file:repository/conf/sample/resources/mocks/GetQuoteResponse.xsl"/>
    <proxy name="StockQuoteService">
        <target>
            <inSequence>
                <xslt key="GetQuoteResponse_xsl"/>
                <property name="RESPONSE" value="true"/>
                <header name="To" action="remove"/>
                <send/>
            </inSequence>
        </target>
    </proxy>
</definitions>
```
In this scenario you defined a proxy service named `StockQuoteService`. A proxy service, as the name suggests, is a virtual service running inside the ESB. It has a definite endpoint location, and you can give a custom WSDL to the service so that this service behaves just like any other Web service hosted at a remote server.

Imagine you're building a client for a remote production service. You can get the WSDL, host it like this, and manipulate the request messages to simulate the real service in minutes. You can even setup WS-Security, WS-ReliableMessaging, and other additions for this service to simulate specific real-world conditions. You can use Synapse to easily add these quality-of-service (QoS) additions to existing production services.

The request messages destined for this service go through the mediators in the `inSequence`. The `inSequence` contents are the same as the contents of the `in` mediator in the previous sample. You replaced the request message with the response message and marked it as a response message.

This configuration doesn't contain a main sequence. You handle only request messages received by the virtual service `StockQuoteService`. If you send the request message to `http://localhost:8081/` as in the previous instance, Synapse logs the message and drops it. That's the default behavior when there's no main sequence.

You've successfully hosted a virtual service in ESB. You're swapping the request message payload with the given response and marking it as a response so that Synapse sends it back, like it's coming from the sample server.

So far you've sent a fixed response message to all the received request messages. Next you see how to customize the response message according to the request message contents.

---

Section 7. Proxy service with a custom response

In this sample, instead of sending a fixed response, you change the response according to the request message contents. As in the previous samples, you use stylesheets to transform the request message into the response message, while
extracting some values from the request message and putting them in the response message.

1. Start Synapse with the synapse_sample_mock3-customquote.xml configuration: ./synapse.sh -sample mock3-customquote.

2. Run the sample client using ant stockquote
   -Dtrpurl=http://localhost:8081/soap/StockQuoteService
   -Dsymbol=ABC.

3. If you monitor the response message, you can see it has the same symbol as the one in the request message. Change the symbol value, and run it again. The response message will contain the given request symbol value.

Listing 4 is the Synapse configuration you used.

Listing 4. Synapse configuration for the proxy service with a custom response: synapse_sample_mock3-customquote.xml

```xml
<definitions xmlns="http://ws.apache.org/ns/synapse">
  <localEntry key="CustomGetQuoteResponse_xsl" src="file:repository/conf/sample/resources/mocks/CustomGetQuoteResponse.xsl"/>
  <proxy name="StockQuoteService">
    <target>
      <inSequence>
        <xslt key="CustomGetQuoteResponse_xsl"/>
        <property name="RESPONSE" value="true"/>
        <header name="To" action="remove"/>
        <send/>
      </inSequence>
      <outSequence/>
    </target>
    <publishWSDL uri="file:repository/conf/sample/resources/mocks/SimpleStockQuoteService.wsdl"/>
  </proxy>
</definitions>
```

Your stylesheet extracts contents from the request message and uses some of it to customize the response. In this case you simply used the `<xsl:value-of/>` element to extract request data and put them in the response message.

Listing 5. Stylesheet for the proxy service with a custom response: CustomGetQuoteResponse.xsl

```xml
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
  <xsl:template match="/"
    soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"">
In this sample you used a stylesheet transformation to build a suitable response message using content extracted from the request message. If you look at the WSDL of this Web service, you can see that there are three more operations in this Web service. At the moment, for any type of request message sent to this proxy service, Synapse is sending the same response. In the next section you implement the other operations of the Web service.

---

**Section 8. Full proxy service**

Now let's implement the full set of operations of the stock quote service using the stylesheet transformations. You're testing for the operation name element in the request message and replacing it with a suitable response.

1. Start Synapse with the synapse_sample_mock4-fullservice.xml configuration: 
   ```bash
   ./synapse.sh -sample mock4-fullservice
   ```

2. Run the sample client as follows: 
   ```bash
   ant stockquote
   -Dtrpurl=http://localhost:8081/soap/StockQuoteService
   -Dmode=quote
   ```

3. Change the `quote` value to `fullquote`, `marketactivity`, and `placeorder` respectively, and run the above command. The sample client sends a different type of request in each case. Synapse sends a suitable response according to the sent request message.

Listing 6 is the Synapse configuration you used.

**Listing 6. Synapse configuration for the full proxy service:**

```xml
<soapenv:Body>
<ns:getQuoteResponse
xmlns:ns="http://services.samples/xsd">
<ns:return
type="samples.services.GetQuoteResponse">
<ns:change>-2.920874334221886</ns:change>
...
<ns:symbol><xsl:value-of
select="ns:getQuote/ns:request/ns:symbol"/></ns:symbol>
<ns:volume>9343</ns:volume>
</ns:return>
</ns:getQuoteResponse>
</soapenv:Body>
</soapenv:Envelope>
</xsl:template>
</xsl:stylesheet>
```
In the stylesheet you're testing what the request message is and replacing the contents with a suitable response message. One exceptional case is the placeOrder operation, which is an in-only operation. That means the placeOrder operation has no response. You used a filter mediator to test whether the request contains the placeOrder element using an XPath expression. If it does have it, you've logged the request and dropped the message, preventing further mediation. This doesn't send a response back to the client as expected for the placeOrder operation.

Listing 7 is the stylesheet used for the full proxy service. You check for elements with operation names and replace the message if any of them are found with a suitable response message.

Listing 7. Stylesheet for the full proxy service: SimpleStockQuoteService.xsl

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                 xmlns:ns="http://services.samples/xsd">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
        <xsl:if test="ns:getQuote">
          <ns:getQuoteResponse>
            ...
        </ns:getQuoteResponse>
        </xsl:if>

        <xsl:if test="ns:getFullQuote">
          <ns:getFullQuoteResponse xmlns:ns="http://services.samples/xsd">
            ...
        </ns:getFullQuoteResponse>
        </xsl:if>
    </soapenv:Body>
</soapenv:Envelope>
</xsl:template>
</xsl:stylesheet>
```
You've successfully simulated the complete Web service operations inside Synapse using a simple stylesheet. In the next section you add another virtual Web service to the existing configuration.

---

**Section 9. Group of services**

In this step you add another virtual Web service, LBService1. LBService1 is another Web service that comes with the bundled sample server in Synapse to demonstrate load balancing and failover samples. Here you simulate that in Synapse alone.

1. Start Synapse with the synapse_sample_mock5-twoproxy.xml configuration: `./synapse.sh -sample mock5-twoproxy`.

2. Run the sample client as follows: `ant loadbalancefailover -Dtrpurl=http://localhost:8081/soap/LBService1 -Di=10`.

The sample client sends a sequence of requests and prints the output from the simulated responses from Synapse. The previous stock quote service is also available. You can try the previous sample requests so you can implement as many services as you need, and you can customize how each one should respond.

You can see the WSDL of the new mock service at `http://localhost:8080/soap/LBService1?wsdl`.

Listing 8 shows the Synapse configuration you used.

---

**Listing 8. Synapse configuration for group of services:**
synapse_sample_mock5-twoproxy.xml

```xml
<ns:getFullQuoteResponse>
</xsl:if>

<xsl:if test="ns:getMarketActivity">
<ns:getMarketActivityResponse
xmlns:ns="http://services.samples/xsd">
...
</ns:getMarketActivityResponse>
</xsl:if>
</soapenv:Body>
</soapenv:Envelope>
</xsl:stylesheet>
```
For LBService1, instead of XSLT mediator you used the script mediator. The script mediator executes scripts in JavaScript, Groovy, Ruby, Python, or any other scripting language that's supported by the scripting engine. You can use the scripting to manipulate the message. In this case I've written a simple code snippet in JavaScript to replace the message payload with a new response. You can even verify the contents of the request using scripting and return a fault if the request is invalid, like a true mock Web service.

In this way you can create a group of mock Web services. You can keep these different configurations and organize them the same way the samples are organized. So the next time you test your sales automation system, you can start the Synapse configuration file, which contains the mock Web services required for that application, and begin testing.

---

**Section 10. Simulate mediator**

So far you've used the existing Synapxe tools to simulate Web services. You can
extend Synapse to support your specific requirements using custom mediators and
tasks, such as a simulate mediator.

The simulate mediator uses a configuration file, as shown in Listing 9. It lists the
regular expression patterns to look for in the incoming message and the file to return
if the set of patterns are matched. You can also give a default file name if none of
the entries are matched. This configuration file is reloaded periodically from the file
system so you can modify this file without restarting Synapse. Using this mediator,
you can instruct Synapse to look for the given patterns in the request message and
send back a specific response. For example, you can tell Synapse to send one
response if the request has `getQuote` and `ABC` text patterns and send a fault
message when the request contains `getQuote` and `DEF` text patterns. Using the
attribute `payload-only`, you can tell the mediator whether to replace the whole
message or to replace only the body contents. Listing 9 is a sample simulate
mediator configuration.

**Listing 9. Simulate mediator configuration**

```xml
<simulate-config>
  <match file='abc.xml'>
    <pattern>urn:getQuote</pattern>
    <pattern>ABC</pattern>
  </match>
  <match file='fault.xml' payload-only='true'>
    <pattern>urn:getQuote</pattern>
    <pattern>[Cc]on</pattern>
  </match>
  <default-file>default.xml</default-file>
</simulate-config>
```

Listing 10 shows the Synapse configuration that demonstrates the simulate mediator
in action. It's similar to the previous samples. However, instead of a transformation,
the simulate mediator replaces the message using the file chosen by evaluating the
simulate mediator configuration listed previously.

**Listing 10. Synapse configuration for using the simulate mediator**

```xml
<definitions xmlns="http://ws.apache.org/ns/synapse">
  <in>
    <simulate>
      <simulate-config>
        repository/conf/sample/resources/simulate/simulate-config.xml</simulate-config>
      <file-path>repository/conf/sample/resources/simulate/</file-path>
    </simulate>
    <property name="RESPONSE" value="true"/>
    <header name="To" action="remove"/>
    <send/>
  </in>
</definitions>
```

You can download the simulate mediator extension from the ESB community site
and try its sample (see Resources for the link). You can also download the source
code and customize it or improve on it.

Section 11. Send requests

Developing and testing the client side without having access to the server side is the hard part. In all of the previous scenarios, you simulated that. But what about building the server-side components? You might have used a tool like `curl` to send a request payload to the endpoint or written a custom client program to call the new Web service you're developing. You can simulate that with Apache Synapse, too.

Listing 11 is the sample 254 in the Apache Synapse distribution with some modifications (see Resources for the link). You've removed the WSDL entry altogether and pointed the endpoint to your testing service. Create the request messages, and copy them into the `in` directory. Apache Synapse picks them from the `in` directory, sends them to the target service, and saves the response in the `out` directory. In this case Synapse acts like a ball launcher, picking up files from the `in` directory, shooting at the given endpoint, and saving the responses in the `out` directory.

Listing 11. Synapse configuration for sending requests

```xml
<definitions xmlns="http://ws.apache.org/ns/synapse">
    <proxy name="StockQuoteProxy" transports="vfs">
        <parameter name="transport.vfs.FileURI">file:///home/upul/test/in</parameter>
        <parameter name="transport.vfs.ContentType">text/xml</parameter>
        <parameter name="transport.vfs.FileNamePattern">.*\.xml</parameter>
        <parameter name="transport.PollInterval">2</parameter>
        <parameter name="transport.vfs.MoveAfterProcess">file:///home/upul/test/original</parameter>
        <parameter name="transport.vfs.MoveAfterFailure">file:///home/upul/test/original</parameter>
        <parameter name="transport.vfs.ActionAfterProcess">MOVE</parameter>
        <parameter name="transport.vfs.ActionAfterFailure">MOVE</parameter>

        <target>
            <endpoint format="soap12" url="http://localhost:9000/soap/TestService"/>
        </endpoint>

        <outSequence>
            <property name="transport.vfs.ReplyFileName" expression="fn:concat(fn:substring-after(get-property('MessageID'), 'urn:uuid:'), '.xml')" scope="transport"/>
        </outSequence>
    </proxy>
</definitions>
```
Section 12. Conclusion

Apache Synapse is a simple, lightweight, and versatile ESB that you can use for much more than enterprise integration development. In this tutorial, you learned how you can use it to simulate and mock Web services by simply transforming, manipulating, and redirecting the request messages.

I’ve barely scratched the surface of what you can do with this multiuse tool. You can write custom mediators and tasks to extend Apache Synapse functionality. (See the articles in the Resources section to learn how to write custom mediators and tasks to fulfill your specific requirements.) If you come up with new ideas and ways to use Apache Synapse to solve large-scale enterprise integration problems or your own personal one-time tasks, please share them with the Apache Synapse ESB community.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample files for this tutorial</td>
<td>ws-synapse_sample_files.zip</td>
<td>9KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

**Information about download methods**
Resources

Learn

• Read the Apache Synapse ESB - Quick Start Guide to learn how to install Apache Synapse and start with the basic examples.

• Read the Apache Synapse samples guide, and try out all the samples.

• Refer to the Apache Synapse configuration language reference guide for comprehensive coverage of the domain-specific language used for Apache Synapse configuration.

• Read Writing a Mediator in WSO2 ESB - Part I to learn how to write a simple mediator to extend Apache Synapse functionality.

• Read Writing a Mediator in WSO2 ESB - Part 2 to learn how to support domain-specific language constructs in your custom mediator.

• Read Writing a Task in WSO2 ESB to learn how to support scheduled jobs in Apache Synapse.

• Check out the ESB community site to download the simulate mediator and other extensions, and learn more about the open source ESB, Apache Synapse.

• The SOA and Web services zone on IBM developerWorks hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.

• Play in the IBM SOA Sandbox! Increase your SOA skills through practical, hands-on experience with the IBM SOA entry points.

• The IBM SOA Web site offers an overview of SOA and how IBM can help you get there.

• Stay current with developerWorks technical events and webcasts.

• Browse for books on these and other technical topics at the Safari bookstore.

• Check out a quick Web services on demand demo.

Get products and technologies

• Download Apache Synapse and try out the samples.

• Download Apache TCPMon to monitor messages between the sample client and Synapse.

• You need Apache Ant to run the sample client.

• Innovate your next development project with IBM trial software, available for download or on DVD.
Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

Upul Godage
Upul Godage works as a senior software engineer at WSO2. He's experienced in enterprise Java, SOA, open source technologies, and Linux. He's involved in Apache Web Services Project, Apache Axis2, and Apache Synapse ESB.

Trademarks

IBM, the IBM logo, and developerWorks are registered trademarks of IBM in the United States, other countries or both.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.