Mocking Web Services

How do you test online services... when you’re not online?

Scott Davis
Davisworld Consulting
Introduction

- My name is Scott Davis
  - **JBoss At Work**
    (O’Reilly)
  - **Google Maps API**
    (Pragmatic Bookshelf)
  - **Pragmatic GIS**
    (Pragmatic Bookshelf)
We’re going to talk about how to test Web Services

- SOAP
- REST
- JSON

A big part of testing Web Services is mocking them up

- A mock is a “stand-in” for the real service
Why Mock?

There are several reasons why you might not want to beat on the real thing during testing:

- Metered service / $$$
- Bandwidth / Lag-time
- Offline testing
  - Laptop at 20,000 feet
Let’s get started by mocking up the most popular of the bunch...

➢ SOAP
What Is SOAP?

- SOAP is one of the most common WS implementations

SOAP is a protocol for exchanging XML-based messages over a computer network, normally using HTTP. SOAP forms the foundation layer of the web services stack, providing a basic messaging framework that more abstract layers can build on. SOAP facilitates the Service-Oriented architectural pattern.

The name "SOAP" was originally an acronym for Simple Object Access Protocol, but the full name was dropped in Version 1.2 of the SOAP specification...

SOAP Specifics

- Transport:
  - Commonly HTTP POST, but can also be SMTP, JMS, etc.

- Request format:
  - XML (SOAP Envelope, Header, Body)

- Response Format
  - XML (SOAP Envelope, Header, Body)
What Is WSDL?

- SOAP is the message format
- WSDL describes the WS interface

The Web Services Description Language (WSDL) is an XML format published for describing Web services.

WSDL describes the public interface to the web service. This is an XML-based service description on how to communicate using the web service; namely, the protocol bindings and message formats required to interact with the web services listed in its directory. The supported operations and messages are described abstractly, and then bound to a concrete network protocol and message format.

(Source: http://en.wikipedia.org/wiki/WSDL)
To see SOAP in action, let’s use the free Google WS API:

With the Google Web APIs service, software developers can query billions of web pages directly from their own computer programs. Google uses the SOAP and WSDL standards so a developer can program in his or her favorite environment – such as Java, Perl, or Visual Studio .NET.

To start writing programs using Google Web APIs:
1. Download the developer’s kit
2. Create a Google Account
3. Write your program using your license key

(Source: [http://www.google.com/apis/](http://www.google.com/apis/))
What we’re trying to accomplish is performing this query programmatically:

- XML results instead of HTML
The WSDL (in the developer’s toolkit we downloaded) describes the WS

The URL

```xml
<!-- Endpoint for Google Web APIs -->
<service name="GoogleSearchService">
  <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
    <soap:address location="http://api.google.com/search/beta2"/>
  </port>
</service>
```
The available method calls:

```xml
<!-- Port for Google Web APIs, "GoogleSearch" -->
<portType name="GoogleSearchPort">

  <operation name="doGetCachedPage">
    <input message="typens:doGetCachedPage"/>
    <output message="typens:doGetCachedPageResponse"/>
  </operation>

  <operation name="doSpellingSuggestion">
    <input message="typens:doSpellingSuggestion"/>
    <output message="typens:doSpellingSuggestionResponse"/>
  </operation>

  <operation name="doGoogleSearch">
    <input message="typens:doGoogleSearch"/>
    <output message="typens:doGoogleSearchResponse"/>
  </operation>

</portType>
```
The binding:

```xml
<binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
  <soap:binding style="rpc">
    <transport>http://schemas.xmlsoap.org/soap/http/</transport>
    <operation name="doGoogleSearch">
      <soap:operation soapAction="urn:GoogleSearchAction"/>
      <input>
        <soap:body use="encoded">
          <urn:GoogleSearch/>
          <encodingStyle>http://schemas.xmlsoap.org/soap/encoding/</encodingStyle>
        </soap:body>
      </input>
      <output>
        <soap:body use="encoded">
          <urn:GoogleSearch/>
          <encodingStyle>http://schemas.xmlsoap.org/soap/encoding/</encodingStyle>
        </soap:body>
      </output>
    </operation>
  </soap:binding>
</binding>
```
The request arguments:

```xml
<message name="doGoogleSearch">
  <part name="key"    type="xsd:string"/>
  <part name="q"      type="xsd:string"/>
  <part name="start"  type="xsd:int"/>
  <part name="maxResults" type="xsd:int"/>
  <part name="filter"  type="xsd:boolean"/>
  <part name="restrict" type="xsd:string"/>
  <part name="safeSearch" type="xsd:boolean"/>
  <part name="lr"     type="xsd:string"/>
  <part name="ie"     type="xsd:string"/>
  <part name="oe"     type="xsd:string"/>
</message>
```
The response format:

```xml
<xs:complexType name="ResultElement">
  <xs:all>
    <xs:element name="summary" type="xs:string"/>
    <xs:element name="URL" type="xs:string"/>
    <xs:element name="snippet" type="xs:string"/>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="cachedSize" type="xs:string"/>
    <xs:element name="relatedInformationPresent" type="xs:boolean"/>
    <xs:element name="hostName" type="xs:string"/>
    <xs:element name="directoryCategory" type="typens:DirectoryCategory"/>
    <xs:element name="directoryTitle" type="xs:string"/>
  </xs:all>
</xs:complexType>
```
Here is the SOAP Request:

```xml
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearch
        xmlns:ns1="urn:GoogleSearch"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    <key xsi:type="xsd:string">000000000000000000000000000000000000000000</key>
    <q xsi:type="xsd:string">jboss at work</q>
    <start xsi:type="xsd:int">0</start>
    <maxResults xsi:type="xsd:int">10</maxResults>
    <filter xsi:type="xsd:boolean">true</filter>
    <restrict xsi:type="xsd:string"></restrict>
    <safeSearch xsi:type="xsd:boolean">false</safeSearch>
    <lr xsi:type="xsd:string"></lr>
    <ie xsi:type="xsd:string">latin1</ie>
    <oe xsi:type="xsd:string">latin1</oe>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
And here is the SOAP Response:

```xml
<?xml version='1.0' encoding='UTF-8'?>

  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
      <return xsi:type="ns1:GoogleSearchResult">
        <documentFiltering xsi:type="xsd:boolean">false</documentFiltering>
        <estimatedTotalResultsCount xsi:type="xsd:int">3</estimatedTotalResultsCount>
        <directoryCategories xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
          xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[0]"></directoryCategories>
        <searchTime xsi:type="xsd:double">8.194871</searchTime>
        <resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
          xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[3]">
          ...
        </resultElements>
      </return>
    </ns1:doGoogleSearchResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
...and the first response element:

```xml
12   <item xsi:type="ns1:ResultElement">
13                    <cachedSize xsi:type="xsd:string">12k</cachedSize>
14                    <hostName xsi:type="xsd:string"></hostName>
15  
16  
17                    <snippet xsi:type="xsd:string">In this first article of a series, you'll learn how
to download and install JBoss. Tom Marrs and Scott Davis, authors of the upcoming JBoss At
Work: A ...</snippet>
18                    <directoryCategory xsi:type="ns1:DirectoryCategory">
19                        <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
20                    <directoryTitle xsi:type="xsd:string"></directoryTitle>
21                    <summary xsi:type="xsd:string"></summary>
22                      <URL xsi:type="xsd:string">http://today.java.net/pub/a/today/2005/03/01/InstallingJBoss.html</URL>
23                
24        
25                    <title xsi:type="xsd:string">&lt;b&gt;JBoss At Work&lt;/b&gt;</title>
26    </item>
```
Before we can mock up the Google Web Service, we need to get our hands on some real output

- SOAPClient4XG
- cURL
- Apache JMeter
- Apache AXIS TCPMonitor
- **SOAPClient4XG**
  - “SOAP Client for XML Geeks”
    - 100 lines of code
    - Allows you to make SOAP requests from the command line
SOAPClient4XG

java -jar sc4xg.jar SC4XG http://api.google.com/search/beta2 sampleSearch.xml
- **cURL**
  - [http://curl.haxx.se/download.html](http://curl.haxx.se/download.html)
  - The ubiquitous command-line tool for performing HTTP GETs and POSTs is available for every platform
cURL

curl --request POST --header "Content-Type: text/xml"
   --data @sampleSpelling.xml http://api.google.com/search/beta2
Apache Jmeter

- This load/stress testing tool has built-in support for SOAP
JMeter Request

File with SOAP XML Data (overrides above text)

Note: Parsing XML is CPU intensive. Therefore, do not set the thread count too high. In general, 10 threads will consume 100% of the CPU on a 300mhz Pentium II. On a pentium 4 2.4ghz cpu, 50 threads is the upper limit. Your options for increasing the number of clients is to increase the number of machines or use multi-cpu systems.
JMeter Response

View Results Tree

Name: View Results Tree

Write All Data to a File

Filename

Sampler result  |  Request  |  Response data

SOAP-ENV:Envelope
  xmlns:soap-ENV = "http://schemas.xmlsoap.org/soap/Envelope/
  xmlns:xsd = "http://www.w3.org/1999/XMLSchema"
  xmlns:xs = "http://www.w3.org/1999/XMLSchema-instance"

SOAP-ENV:Body
  ns1:doSpellingSuggestionResponse
    SOAP-ENV:encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/
    xmlns:ns1 = "urn:GoogleSearch"
    return
      xsi:type = "xsd:string"
      pcato
Apache AXIS TCPMonitor

- http://ws.apache.org/axis/
- You might already be using this as your SOAP service provider, but there is a hidden gem buried deep inside...
TCPMonitor

java -classpath $AXIS_HOME/lib/axis.jar org.apache.axis.utils.tcpmon

curl --request POST --header "Content-Type: text/xml"
   --data @sampleSpelling.xml http://localhost:8989/search/beta2

SOAP Request

URL to SOAP service
(Notice the URL points to localhost:8989)
TCPMonitor Setup

Create a new TCP/IP Monitor...

Listen Port #: 8089

Act as a...

- **Listener**
  - Target Hostname: api.google.com
  - Target Port #: 80

- **Proxy**

Options

- HTTP Proxy Support

- Simulate Slow Connection
  - Bytes per Pause
  - Delay in Milliseconds

**Add**
TCPMonitor Response

--- Most Recent ---
Done: 2006-08-30 23:08:10
localhost api.google.com
POST /search/beta2 HTTP/1.1 ...

HTTP/1.1 200 OK
Content-Type: text/xml; charset=UTF-8
Cache-control: private
Transfer-Encoding: chunked
Date: Thu, 31 Aug 2006 05:08:02 CMT
Server: CFF/1.3

<?xml version="1.0" encoding="UTF-8"?>
<ns1:doSpellingSuggestionResponse xmlns:ns1="urn:Google">
<return xsi:type="xsd:string">potato</return>
</ns1:doSpellingSuggestionResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Now that we have some valid output, how can we set up a mock Google web service?

- JSP
- Groovlets
JSP

- Gives you the flexibility to simply copy/paste in the XML/SOAP response

```xml
<% response.setContentType("text/xml"); %>

  .envelope/" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">potato</return>
    </ns1:doSpellingSuggestionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Groovy

- Groovy is a scripting language that runs atop Java
  - You can run it as an interpreted (uncompiled) scripting language
  - You can compile it down to bytecode and mix it in with your existing Java codebase
    - [http://groovy.codehaus.org](http://groovy.codehaus.org)
**Groovlets = Groovy + Servlets**

- Once you add `groovy.servlet.GroovyServlet` to `web.xml`, you can add Groovy scripts on the fly:
  - No redeploy necessary
  - `GroovyServlet` compiles and runs the scripts automagically
```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd">

  <servlet>
    <servlet-name>Groovy</servlet-name>
    <servlet-class>groovy.servlet.GroovyServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Groovy</servlet-name>
    <url-pattern>*.groovy</url-pattern>
  </servlet-mapping>

  <!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```
WEB-INF/lib

- Just add a few choice JARs to WEB-INF/lib...

```
rosencrantz:/Library/tomcat/webapps/groovlets/WEB-INF/lib sdavis$ ls -al
total 3960
drwxr-xr-x  6 sdavis admin 204 May 24 23:48 .
drwxr-xr-x  4 sdavis admin 136 May 24 23:46 ..
-rw-r--r--  1 sdavis admin 435563 May 24 23:48 antlr-2.7.5.jar
-rw-r--r--  1 sdavis admin  34778 May 24 23:48 asm-2.2.jar
-rwxr-xr-x  1 sdavis admin 295948 May 24 23:46 derbyclient.jar
-rw-r--r--  1 sdavis admin 1249663 May 24 23:46 groovy-1.0-jsr-05.jar
```
soap.groovy

```groovy
xml = """"<?xml version='1.0' encoding='UTF-8'?>
 . xmlns:ns1="urn:GoogleSearch"
 . xmlns:xsd="http://www.w3.org/1999/XMLSchema-instance"
 . xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
 . SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">potato</return>
    </ns1:doSpellingSuggestionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>"

response.setContentType("text/xml")
println xml
```

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <SOAP-ENV:Envelope>
  - <SOAP-ENV:Body>
    - <ns1:doSpellingSuggestionResponse SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">potato</return>
    </ns1:doSpellingSuggestionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
While the JSP and Groovlet examples look virtually identical at first glance, the Groovlet is far more robust.

You have the full programming language at your fingertips.
```xml
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="xsd:string">
      ""
    </ns1:doSpellingSuggestionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Value = request.getParameter('return')

if (value == null) { value = "" }

response.setContentType("text/xml")

println "${xmlStart}${value}${xmlEnd}"

This XML file does not appear to have any style information associated with it. The document tree is shown below.

- <SOAP-ENV:Envelope>
  - <SOAP-ENV:Body>
    - <ns1:doSpellingSuggestionResponse SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">foo</return>
    </ns1:doSpellingSuggestionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Now that we have a mock Google web service, how can we call it?

- DNS trickery
- WSDL trickery
DNS Trickery

- Add “127.0.0.1 api.google.com” to /etc/hosts
- Create a “search” webapp
- Create a “beta2” subdirectory
  ➢ Or do sneaky URL redirects using TCPMon
WSDL Trickery

- The WSDL (in the developer’s toolkit we downloaded) describes the WS

  The URL

```xml
<service name="GoogleSearchService">
  <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
    <soap:address location="http://api.google.com/search/beta2"/>
  </port>
</service>
```

- Flip this to http://localhost
SOAP Conclusion

- SOAP is only one implementation of WS.
  - There is a “kinder, gentler” form of WS available to you
What Is REST?

The other white meat:

Representational State Transfer (REST) is a software architectural style for distributed hypermedia systems like the world wide web. The term originated in a 2000 doctoral dissertation about the web written by Roy Fielding, one of the principal authors of the HTTP protocol specification, and has quickly passed into widespread use in the networking community.

Systems that follow Fielding's REST principles are often referred to as RESTful; REST's most zealous advocates call themselves RESTafarians.

(Source: http://en.wikipedia.org/wiki/Representational_State_Transfer)
There are two “types” of RESTful WS out there:

- Pure REST (following Fielding’s principles)
- Popular REST (essentially what all of the mainstream websites provide)
Popular REST is a reaction to the complexity of SOAP

- Transport: HTTP GET
- Request format: URL + QueryString
- Response format: XML

• Sometimes it is informally called POX (Plain Old XML) in homage to POJOs (Plain Old Java Objects).
To see a “Popular” RESTful WS in action, let’s go over to Yahoo!

How do I get started?

1. Online Documentation
   Read the online documentation and FAQs.

2. Get an Application ID
   To access Yahoo! Search Webservices, you will need to get an application ID. Like a browser’s User-Agent string, the Application ID uniquely identifies your application and has no effect on rate limiting.

3. Download the SDK
   The development kit includes BSD licensed examples and libraries for various languages: Perl, Python and PHP, Java, JavaScript, and Flash.

(Source:  http://developer.yahoo.net/search/index.html)
Yahoo offers close to 20 different RESTful services (Search, Maps, Weather, Traffic, Finance, Shopping)

Here is the same web search we did in Google:

POX

- <ResultSet xsi:schemaLocation="urn:yahoo:src http://api.search.yahoo.com/WebSearchService"/>
  - <Result>
    - <Title>
      Amazon.com: JBoss at Work: A Practical Guide: Books
    </Title>
    - <Summary>
      Amazon.com: JBoss at Work: A Practical Guide: Books by Tom Marrs, Scott Davis ... wh
    </Summary>
    - <Url>
    </Url>
    - <ClickUrl>
      http://uk.wrs.yahoo.com/_ylt=A9h7fSGYz_pDVWAATqDdmMwF;_ylu=X3oDMTBjC
    </ClickUrl>
    <ModificationDate>1137312000</ModificationDate>
    <MimeType>text/html</MimeType>
  - <Cache>
    - <Url>
      http://uk.wrs.yahoo.com/_ylt=A9h7fSGYz_pDVWAATqDdmMwF;_ylu=X3oDMTA
    </Url>
    <Size>112159</Size>
  </Result>
How do we get valid sample output?

All of the tools we examined earlier, plus:

- The Browser
- wget
The Browser

This XML file does not appear to have any style information associated with it. The document tree is shown below.

- <ResultSet xsi:schemaLocation="urn:yahoo:srch http://api.search.yahoo.com/WebSearchService/V1/WebSearchResponse.xsd" type="web" totalResultsAvailable="3200000">
  - <Result>
    <Title>JBoss at Work: A Practical Guide</Title>
    <Summary>
      from O'Reilly takes an in-depth look at JBoss, the fastest growing open source tool on the market. Learn how to integrate JBoss with other open source Java products such as
    </Summary>
    <Url>http://www.jbossatwork.com/</Url>
    <ClickUrl>http://uk.wrs.yahoo.com/_ylt=A9bby5zRi_ZEUTkBmzodmMwF_ylu=X3oDMTB2b2gxdDdtBGNvbG8DZQRsA1dTMQRwb3MDMQRzZWMDc3IEdnRpfZAM-/SIG=15t46k</ClickUrl>
    <DisplayUrl>www.jbossatwork.com</DisplayUrl>
    <ModificationDate>1130396400</ModificationDate>
    <MimeType>text/html</MimeType>
    - <Cache>
      <Url>
        http://uk.wrs.yahoo.com/_ylt=A9bby5zRi_ZEUTkBnIzdmMwF_ylu=X3oDMTBwZTdwBWhkBGNvbG8DZQRwb3MDMQRzZWMDc3IEdnRpfZAM-/SIG=15t46k</Url>
      <Size>6164</Size>
    </Cache>
  </Result>
  - <Result>
    <Title>oreilly.com -- Online Catalog: JBoss at Work: A Practical Guide</Title>
    <Summary>
      O'Reilly takes an in-depth look at JBoss, the fastest growing open source tool ... JBoss at Work: A Practical Guide helps developers overcome these challenges. ...
    </Summary>
    <Url>http://www.oreilly.com/catalog/jboss/</Url>
    <ClickUrl>http://uk.wrs.yahoo.com/_ylt=A9bby5zRl_ZEUTkBnIzdmMwF_ylu=X3oDMTB2dnY0Nn1BGNvbG8DZQRsA1dTMQRwb3MDMGRzZWMDc3IEdnRpfZAM-/SIG=15t46k</ClickUrl>
    <DisplayUrl>www.oreilly.com/catalog/jboss/</DisplayUrl>
  </Result>
</ResultSet>
- **wget**
  - Does basically the same thing as cURL, only the syntax is a bit simpler for GETs
wget –O out.xml
query=jboss+at+work"
Many popular websites offer both SOAP and RESTful interfaces

- Amazon
- eBay
  - [http://developer.ebay.com/common/api](http://developer.ebay.com/common/api)

A good search term to discover WS offerings is “[website] api”

- Ebay api
- Amazon api
Amazon mocking
  
  http://awszone.com/

Amazon has a much more complicated RESTful API
  
  Awszone helps tremendously
  
  Once you have the requests built and the sample output, mocking it up is a breeze
AWS Zone - Amazon Web Services (TM) - Made Simple. AWS Code Samples, Code Generator REST and SO...

AWS Zone - Amazon Web Services (TM) - Made Simple. AWS Code Samples, Code Generator REST and SO...

AWS zone.com

Scratch Pads

Amazon S3

Amazon E-Commerce Service

API Version: 2005-03-23

US - United States

BrowseHotelLookup
CartAdd
CartCreate
CartClear
CartGet
CartModify
CustomerContentLookup
CustomerContentSearch
Help
ItemLookup
ItemSearch
ListLookup
ListSearch
SellerListingLookup
SellerListingSearch
SellerLookup
SimilarlyLookup
TransactionLookup

Done

AWS Resources

What's New in AWS
AWS Blog
Resource Center
Developer Forums
Solutions Catalog
AWS ItemSearch
AWS Code Generator

```java
package com.amazonaws.samples.AWSECommerceService;

import com.amazonaws.proxy.AWSECommerceService.*;
import org.apache.axis.types.*;
import java.math.BigDecimal;

public class ItemSearchSample {

    /**
     * Main - API demo
     * @param args
     */
    public static void main ( String[] args )
    {
        try {
            AWSECommerceService locator = new AWSECommerceServiceLocator();
            AWSECommerceServicePortType service = locator.getAWSECommerceServicePort();

            ItemSearch ItemSearch = new ItemSearch();

            ItemSearchRequest request = new ItemSearchRequest();
            request.setContentType ( "text/xml" );
            request.setSubscriptionId ( "0525E2PQ81DD7ZTWTK82" );
            request.setRequestType ( "Small" );

            responseIndex ( "Books" );
            request.setTitle ( "Boss At Work" );

            // Setup request
            ItemSearch.setRequest ( new ItemSearchRequest () { request } );
        }
    }
```
REST Conclusion

- With the popularity of AJAX on the rise (and the continuing weakness of native browser XML support), there is a third WS “flavor” gaining steam:
What Is JSON?

We don’t need no stinkin’ XML…

JSON (pronounced Jason), which stands for "JavaScript Object Notation", is a lightweight computer data interchange format. JSON is a subset of the object literal notation of JavaScript but its use does not require JavaScript.

JSON's simplicity has resulted in its widespread use, especially as an alternative to XML in Ajax. One of the claimed advantages of JSON over XML as a data interchange format in this context is that it is much easier to write a JSON parser. In JavaScript itself, JSON can be parsed trivially using the eval() procedure. This was important for the acceptance of JSON within the AJAX programming community because of JavaScript's ubiquity among web browsers.

(Source:  http://en.wikipedia.org/wiki/JSON)
- **JSON specifics:**
  - Transport: HTTP GET
  - Request format: URL + QueryString
  - Response format: JSON
The following is a simple example of a menubar definition using JSON and XML data encodings.

**JSON:**

```json
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                {"value": "New", "onclick": "CreateNewDoc()"},
                {"value": "Open", "onclick": "OpenDoc()"},
                {"value": "Close", "onclick": "CloseDoc()"}
            ]
        }
    }
}
```

**XML:**

```xml
<menu id="file" value="File">
    <popup>
        <menuitem value="New" onclick="CreateNewDoc()" />
        <menuitem value="Open" onclick="OpenDoc()" />
        <menuitem value="Close" onclick="CloseDoc()" />
    </popup>
</menu>
```
From a JavaScript perspective, once you have a JSON string, all you need to do is `eval()` it:

- `Eval(menuDef);`
- No need for marshalling framework like Castor or XMLBeans

Ruby uses a superset of JSON called YAML

YAML is a recursive acronym meaning "YAML Ain't Markup Language". Early in its development, YAML was said to mean "Yet Another Markup Language", retronymed to distinguish its purpose as data-centric, rather than document markup.

Even though JSON has “JavaScript” in its name, there are bindings for over 15 other programming languages

See [http://www.json.org/](http://www.json.org/) for open source parsers
Getting sample output

wget
Yahoo! Supports JSON output by simply adding another query parameter:

- `output=json`

For more information:
- [http://developer.yahoo.com/common/json.html](http://developer.yahoo.com/common/json.html)
We talked about how to test Web Services

- SOAP
- REST
- JSON

There are many tools that helped us out along the way:

- SOAPClient4XG
- cURL
- Apache Jmeter
- Apache Axis TCPMonitor
- Firefox / Safari / IE / et al
- wget
Conclusion

- Thanks for your time!

- Questions?
  - Email: scottdavis99@yahoo.com
  - Download slides:
    - http://www.davisworld.org/presentations