

# Mocking Web Services

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

---

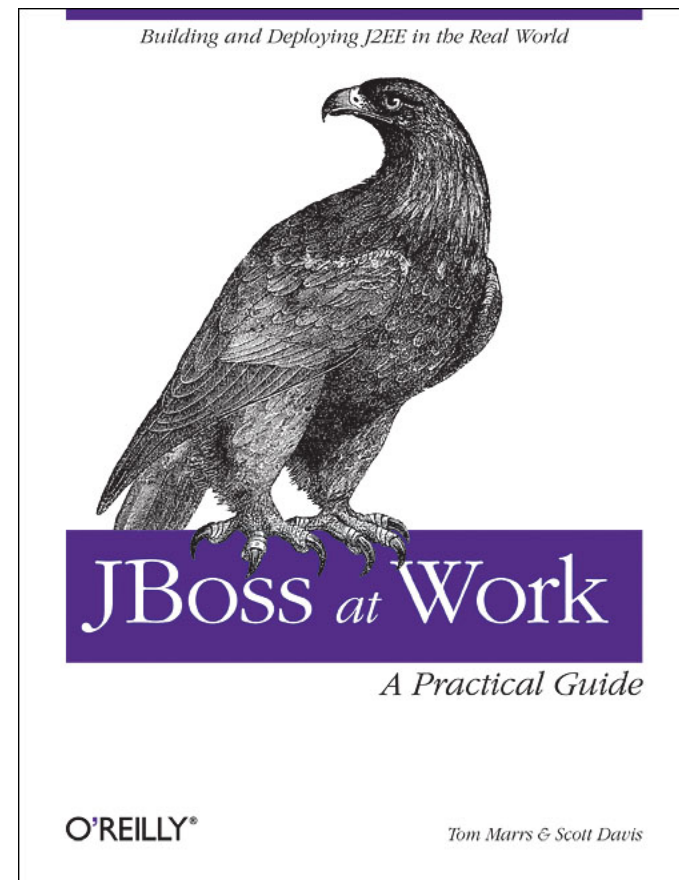
Scott Davis

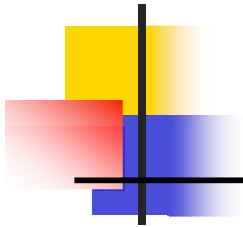
Davisworld Consulting



# Introduction

- My name is Scott Davis
  - JBoss At Work  
(O'Reilly)
  - Google Maps API  
(Pragmatic Bookshelf)
  - Pragmatic GIS  
(Pragmatic Bookshelf)





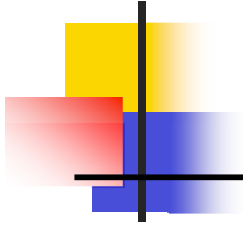
- We're going to talk about how to test Web Services
  - SOAP
  - REST
  - JSON
  
- A big part of testing Web Services is mocking them up
  - A mock is a "stand-in" for the real service



# 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...

(Source: <http://en.wikipedia.org/wiki/SOAP>)



# 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?

---

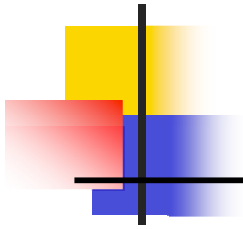
- SOAP is the message format
- WSDL describes the WS interface

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

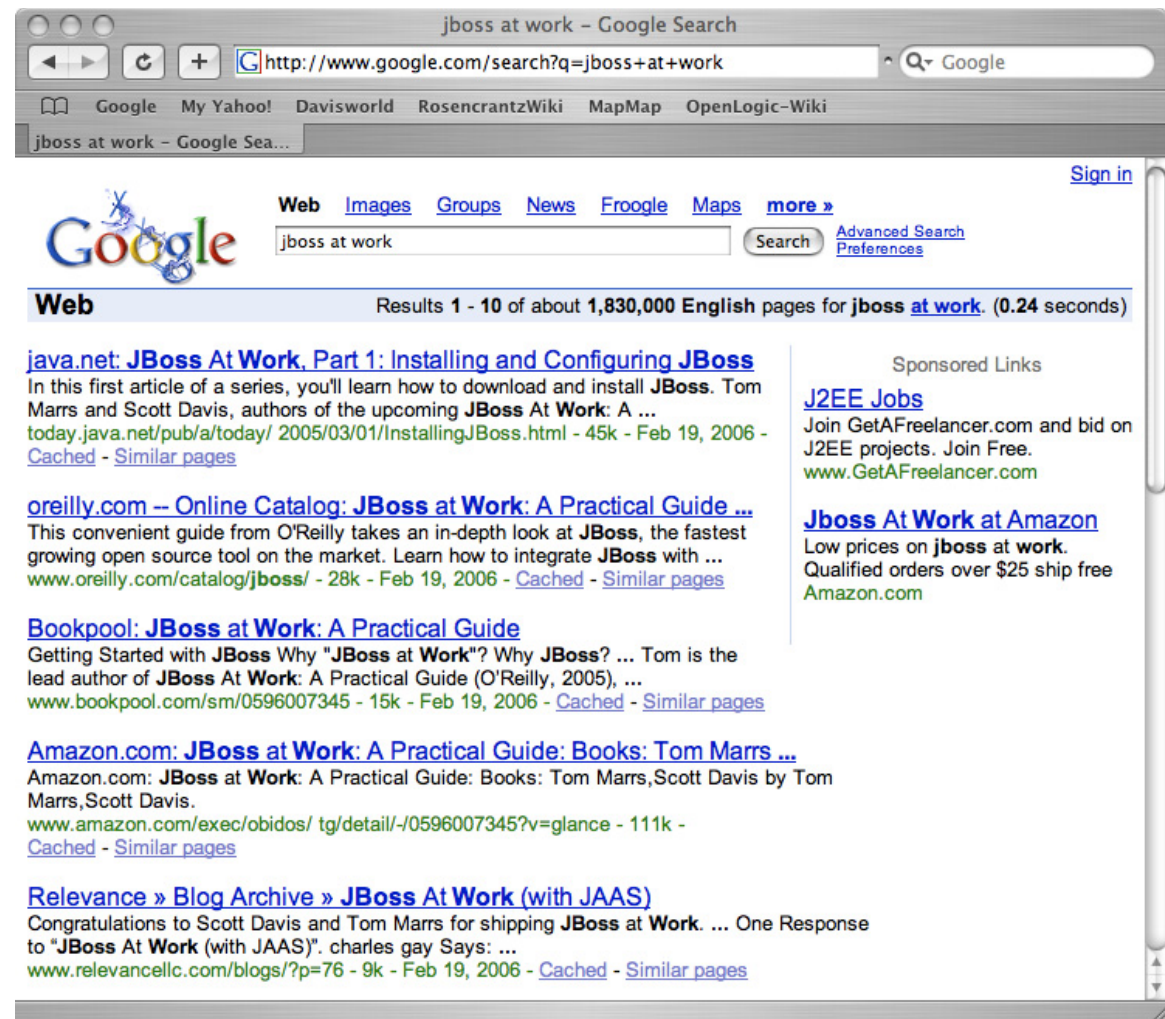
(Source: <http://www.google.com/apis/>)

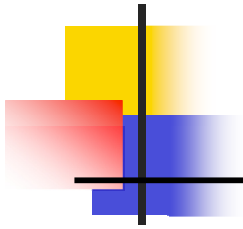


- What we're trying to accomplish is performing this query programmatically

➤ XML results instead of

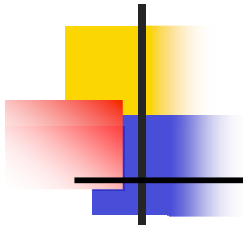
HTML





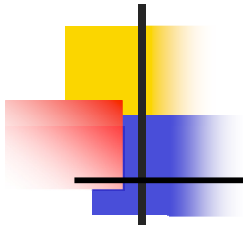
- The WSDL (in the developer's toolkit we downloaded) describes the WS
  - The URL

```
1 <!-- Endpoint for Google Web APIs -->
2 <service name="GoogleSearchService">
3   <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
4     <soap:address location="http://api.google.com/search/beta2"/>
5   </port>
6 </service>
```



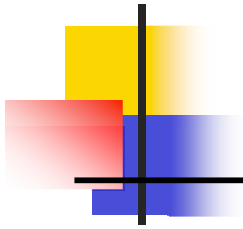
- The available method calls:

```
10 <!-- Port for Google Web APIs, "GoogleSearch" -->
11 <portType name="GoogleSearchPort">
12
13   <operation name="doGetCachedPage">
14     <input message="typens:doGetCachedPage"/>
15     <output message="typens:doGetCachedPageResponse"/>
16   </operation>
17
18   <operation name="doSpellingSuggestion">
19     <input message="typens:doSpellingSuggestion"/>
20     <output message="typens:doSpellingSuggestionResponse"/>
21   </operation>
22
23   <operation name="doGoogleSearch">
24     <input message="typens:doGoogleSearch"/>
25     <output message="typens:doGoogleSearchResponse"/>
26   </operation>
27 </portType>
--
```



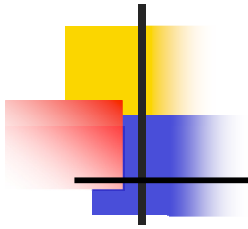
## ■ The binding:

```
34 <!-- Binding for Google Web APIs - RPC, SOAP over HTTP -->
35 <binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
36   <soap:binding style="rpc"
37     transport="http://schemas.xmlsoap.org/soap/http"/>
38   <operation name="doGoogleSearch">
39     <soap:operation soapAction="urn:GoogleSearchAction"/>
40     <input>
41       <soap:body use="encoded"
42         namespace="urn:GoogleSearch"
43         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
44     </input>
45     <output>
46       <soap:body use="encoded"
47         namespace="urn:GoogleSearch"
48         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
49     </output>
50   </operation>
51 </binding>
```



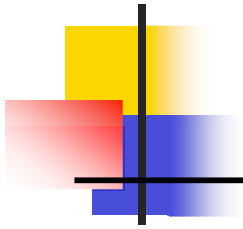
- The request arguments:

```
58 <message name="doGoogleSearch">
59   <part name="key"           type="xsd:string"/>
60   <part name="q"             type="xsd:string"/>
61   <part name="start"         type="xsd:int"/>
62   <part name="maxResults"    type="xsd:int"/>
63   <part name="filter"        type="xsd:boolean"/>
64   <part name="restrict"      type="xsd:string"/>
65   <part name="safeSearch"    type="xsd:boolean"/>
66   <part name="lr"            type="xsd:string"/>
67   <part name="ie"            type="xsd:string"/>
68   <part name="oe"            type="xsd:string"/>
69 </message>
```



## ■ The response format:

```
75 <xsd:complexType name="ResultElement">
76   <xsd:all>
77     <xsd:element name="summary" type="xsd:string"/>
78     <xsd:element name="URL" type="xsd:string"/>
79     <xsd:element name="snippet" type="xsd:string"/>
80     <xsd:element name="title" type="xsd:string"/>
81     <xsd:element name="cachedSize" type="xsd:string"/>
82     <xsd:element name="relatedInformationPresent" type="xsd:boolean"/>
83     <xsd:element name="hostName" type="xsd:string"/>
84     <xsd:element name="directoryCategory" type="typens:DirectoryCategory"/>
85     <xsd:element name="directoryTitle" type="xsd:string"/>
86   </xsd:all>
87 </xsd:complexType>
```



- Here is the SOAP Request:

```

1 <?xml version='1.0' encoding='UTF-8'?>
2
3 <SOAP-ENV:Envelope
4     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
5     xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
6     xmlns:xsd="http://www.w3.org/1999/XMLSchema">
7     <SOAP-ENV:Body>
8         <ns1:doGoogleSearch
9             xmlns:ns1="urn:GoogleSearch"
10             SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
11             <key xsi:type="xsd:string">00000000000000000000000000000000</key>
12             <q xsi:type="xsd:string">jboss at work</q>
13             <start xsi:type="xsd:int">0</start>
14             <maxResults xsi:type="xsd:int">10</maxResults>
15             <filter xsi:type="xsd:boolean">true</filter>
16             <restrict xsi:type="xsd:string"></restrict>
17             <safeSearch xsi:type="xsd:boolean">false</safeSearch>
18             <lr xsi:type="xsd:string"></lr>
19             <ie xsi:type="xsd:string">latin1</ie>
20             <oe xsi:type="xsd:string">latin1</oe>
21         </ns1:doGoogleSearch>
22     </SOAP-ENV:Body>
23 </SOAP-ENV:Envelope>

```







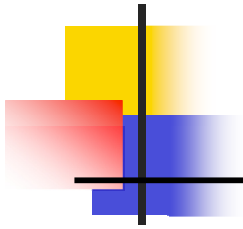
## ■ And here is the SOAP Response:

```
1 <?xml version='1.0' encoding='UTF-8'?>
2
3 <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">
4   <SOAP-ENV:Body>
5     <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" SOAP-ENV:encodingStyle="http://
  . schemas.xmlsoap.org/soap/encoding/">
6       <return xsi:type="ns1:GoogleSearchResult">
7         <documentFiltering xsi:type="xsd:boolean">false</documentFiltering>
8         <estimatedTotalResultsCount xsi:type="xsd:int">3</estimatedTotalResultsCount>
9         <directoryCategories xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
  . xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[0]"></directoryCategories>
10        <searchTime xsi:type="xsd:double">0.194871</searchTime>
11        <resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
  . xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[3]">
```

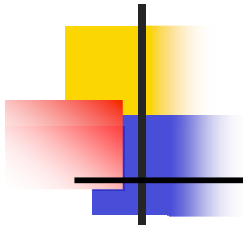


## ■ ...and the first response element:

```
12     <item xsi:type="ns1:ResultElement">
13         <cachedSize xsi:type="xsd:string">12k</cachedSize>
14         <hostName xsi:type="xsd:string"></hostName>
15         <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>
16         <directoryCategory xsi:type="ns1:DirectoryCategory">
17             <specialEncoding xsi:type="xsd:string"></specialEncoding>
18             <fullViewableName xsi:type="xsd:string"></fullViewableName>
19         </directoryCategory>
20         <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
21         <directoryTitle xsi:type="xsd:string"></directoryTitle>
22         <summary xsi:type="xsd:string"></summary>
23         <URL xsi:type="xsd:string">http://today.java.net/pub/a/today/2005/03/01/
.   InstallingJBoss.html</URL>
24         <title xsi:type="xsd:string">&lt;b>JBoss At Work</b></title>
25     </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


- <http://www-128.ibm.com/developerworks/xml/library/x-soapcl/>
- "SOAP Client for XML Geeks"
  - 100 lines of code
  - Allows you to make SOAP requests from the command line



# SOAPClient4XG

---

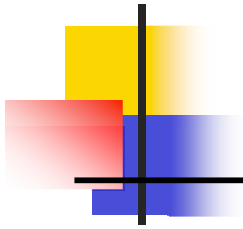
```
java -jar sc4xg.jar SC4XG http://api.google.com/search/beta2 sampleSearch.xml
```



URL to SOAP service



SOAP Request



- cURL

- <http://curl.haxx.se/download.html>
- The ubiquitous command-line tool for performing HTTP GETs and POSTs is available for every platform

grows those URLs

cURL

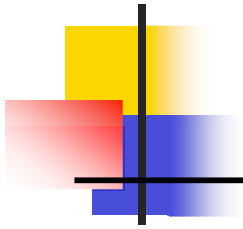


```
curl --request POST --header "Content-Type: text/xml"  
--data @sampleSpelling.xml http://api.google.com/search/beta2
```

SOAP Request

URL to SOAP service





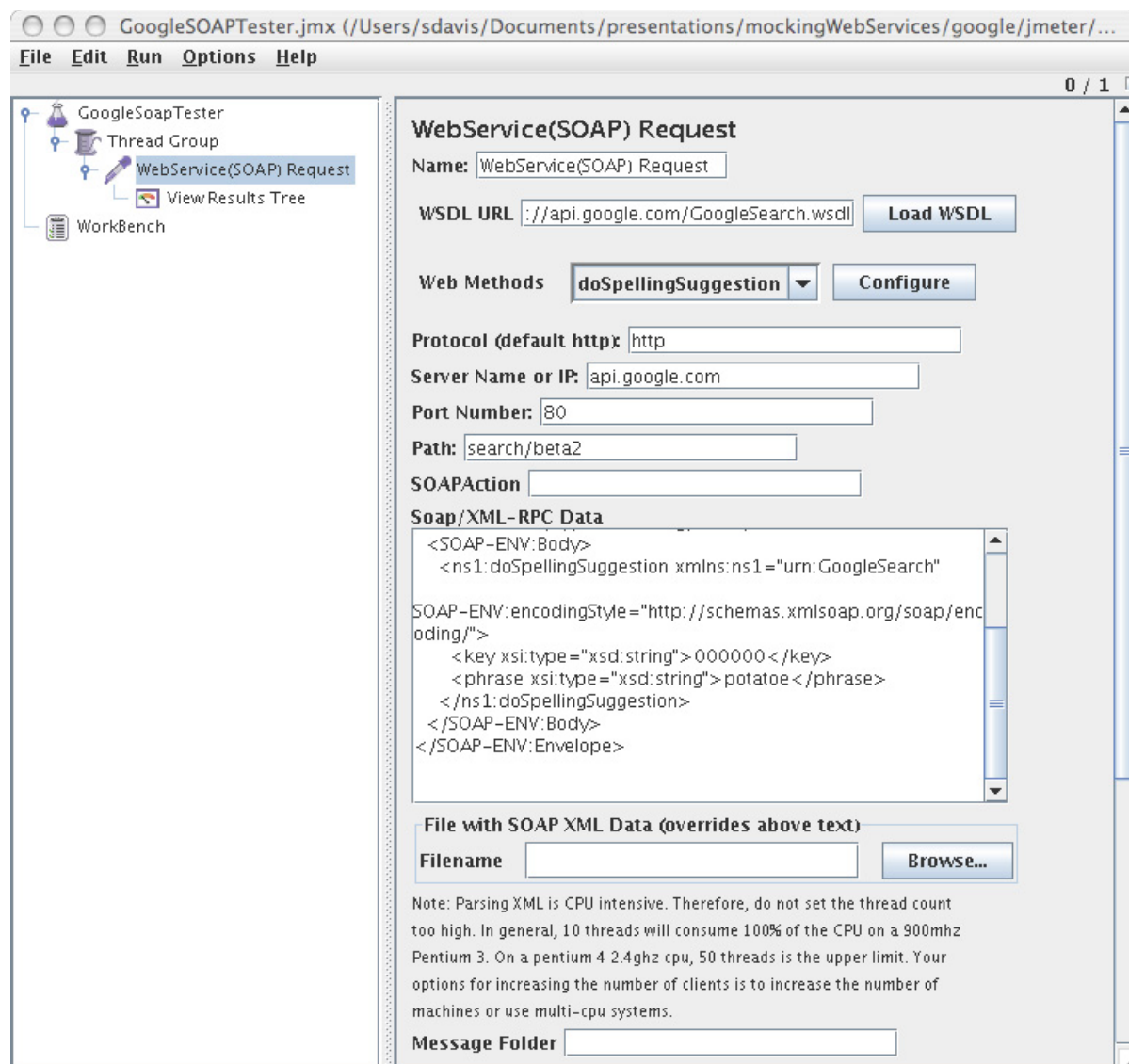
- Apache Jmeter

- <http://jakarta.apache.org/jmeter/>
- This load/stress testing tool has built-in support for SOAP





# JMeter Request



The screenshot shows the JMeter configuration window for a WebService(SOAP) Request. The window title is "GoogleSOAPTester.jmx (/Users/sdavis/Documents/presentations/mockWebServices/google/jmeter/...". The menu bar includes File, Edit, Run, Options, and Help. The left sidebar shows a tree view with GoogleSoapTester, Thread Group, WebService(SOAP) Request (selected), View Results Tree, and WorkBench. The main configuration area is titled "WebService(SOAP) Request" and contains the following fields and buttons:

- Name:** WebService(SOAP) Request
- WSDL URL:**  **Load WSDL**
- Web Methods:** **doSpellingSuggestion** **Configure**
- Protocol (default http):**
- Server Name or IP:**
- Port Number:**
- Path:**
- SOAPAction:**
- Soap/XML-RPC Data:**

```
<SOAP-ENV:Body>
  <ns1:doSpellingSuggestion xmlns:ns1="urn:GoogleSearch"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <key xsi:type="xsd:string">000000</key>
    <phrase xsi:type="xsd:string">potatoe</phrase>
  </ns1:doSpellingSuggestion>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
- File with SOAP XML Data (overrides above text):**  
**Filename:**  **Browse...**
- 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 900mhz Pentium 3. 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.
- Message Folder:**



# JMeter Response

GoogleSOAPTester.jmx (/Users/sdavis/Documents/presentations/mockWebServices/google/jmeter/GoogleSOAPTester.jmx) – Apache JMeter

File Edit Run Options Help

0 / 1

GoogleSoapTester

- Thread Group
  - WebService(SOAP) Request
    - View Results Tree
- WorkBench

**View Results Tree**

Name: View Results Tree

Write All Data to a File

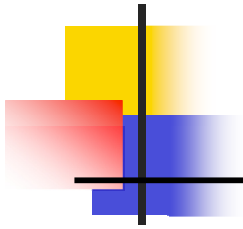
Filename   ☐ Log Errors Only

WebService(SOAP) Request

**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:xsi = "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"
        - potato

☐ Show Text ☐ Render HTML ☒ Render XML ☐ Download embedded resources



- 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...




The logo consists of a black crosshair overlaid on a yellow square, a red square, and a blue square.

# 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

TCPMonitor

Admin

Create a new TCP/IP Monitor...

Listen Port # 8989

Act as a...

☒ Listener

Target Hostname api.google.com

Target Port # 80

☐ Proxy

Options

☐ HTTP Proxy Support

Hostname

Port #

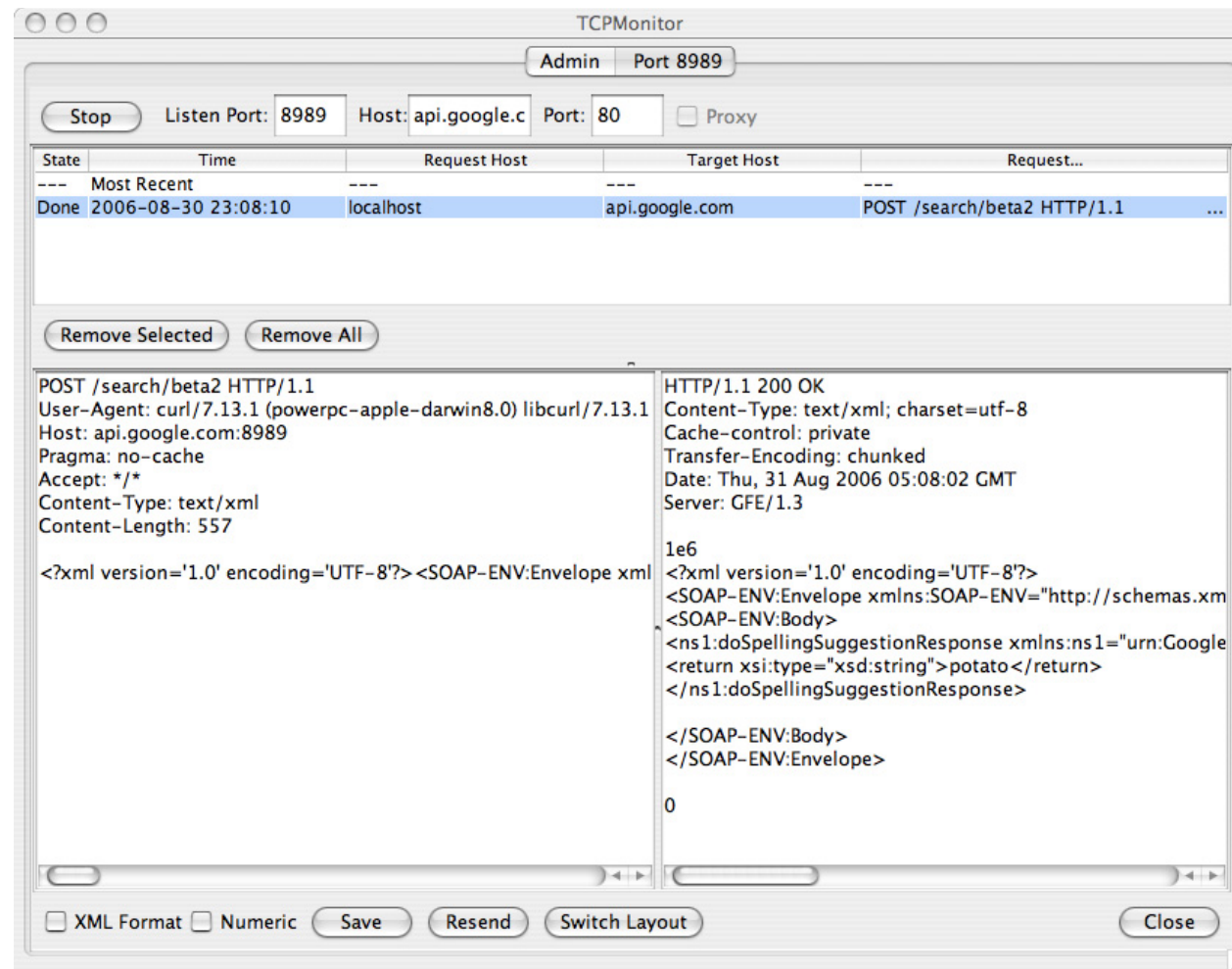
☐ Simulate Slow Connection

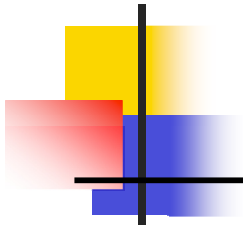
Bytes per Pause

Delay in Milliseconds

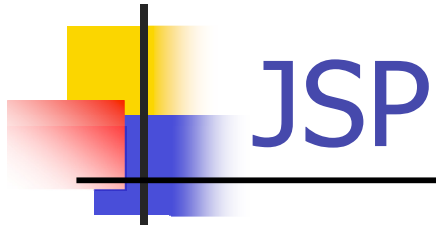
Add

# TCPMonitor Response





- 
- Now that we have some valid output, how can we set up a mock Google web service?
    - JSP
    - Groovlets



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

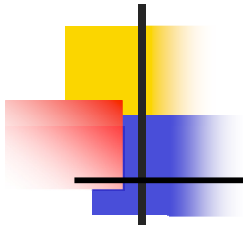
```
1 <% response.setContentType("text/xml"); %>
2
3 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
  . envelope/" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  . xmlns:xsd="http://www.w3.org/1999/XMLSchema">
4   <SOAP-ENV:Body>
5     <ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
  . SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
6       <return xsi:type="xsd:string">potato</return>
7     </ns1:doSpellingSuggestionResponse>
8   </SOAP-ENV:Body>
9 </SOAP-ENV:Envelope>
```





- 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>





- 
- 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



# Web.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <web-app version="2.4"
4      xmlns="http://java.sun.com/xml/ns/j2ee"
5      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd">
7
8      <servlet>
9          <servlet-name>Groovy</servlet-name>
10         <servlet-class>groovy.servlet.GroovyServlet</servlet-class>
11     </servlet>
12
13     <servlet-mapping>
14         <servlet-name>Groovy</servlet-name>
15         <url-pattern>*.groovy</url-pattern>
16     </servlet-mapping>
17
18     <!-- The Welcome File List -->
19     <welcome-file-list>
20         <welcome-file>index.jsp</welcome-file>
21     </welcome-file-list>
22
23 </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
```



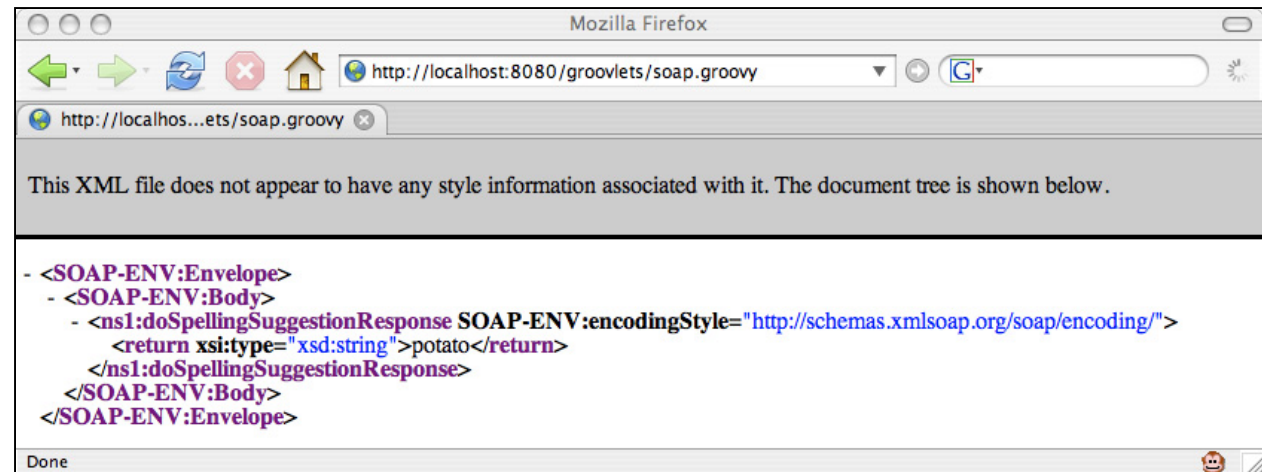


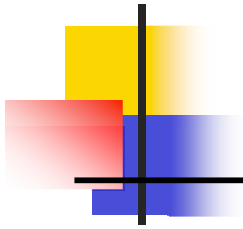
# soap.groovy

```

1 xml = ""<?xml version='1.0' encoding='UTF-8'?>
2 <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">
3   <SOAP-ENV:Body>
4     <ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
  . SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
5       <return xsi:type="xsd:string">potato</return>
6     </ns1:doSpellingSuggestionResponse>
7   </SOAP-ENV:Body>
8 </SOAP-ENV:Envelope>""
9
10 response.setContentType("text/xml")
11 println xml

```





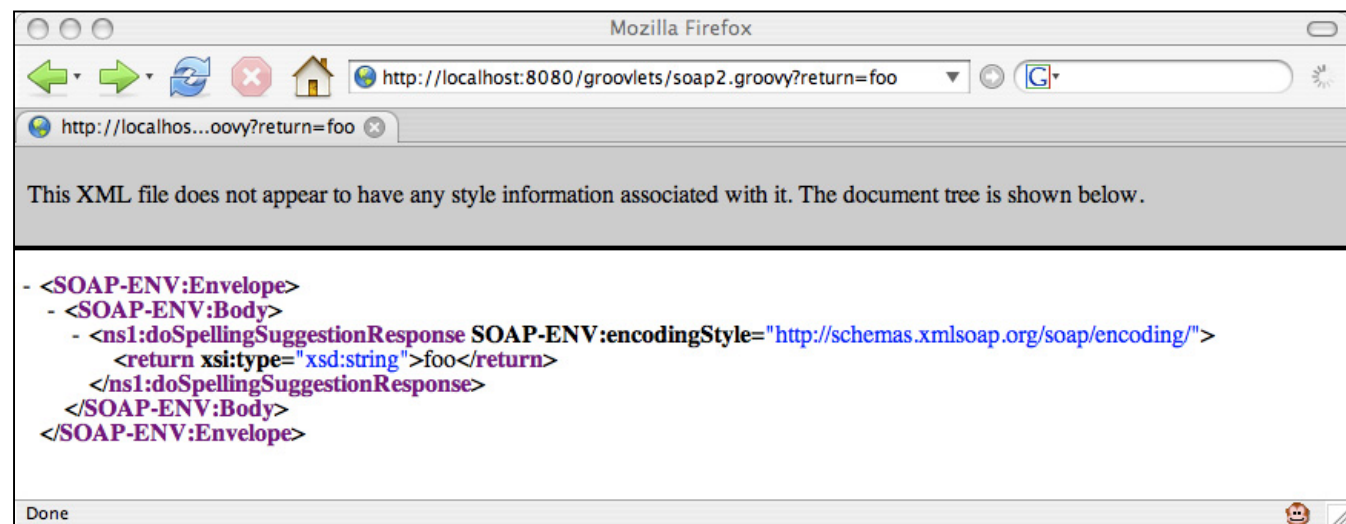
- While the JSP and Groovlet examples look virtually identical at first glance, the Groovlet is far more robust
  - You have the full programming language at your fingertips

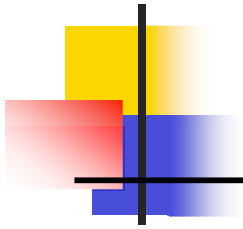


```

1 xmlStart = "<?xml version='1.0' encoding='UTF-8'?"
2 <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">
3   <SOAP-ENV:Body>
4     <ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
. SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
5       <return xsi:type="xsd:string">"
6
7 xmlEnd = "</return>
8       </ns1:doSpellingSuggestionResponse>
9     </SOAP-ENV:Body>
10  </SOAP-ENV:Envelope>"
11
12 value = request.getParameter('return')
13 if (value == null) { value = "" }
14
15 response.setContentType("text/xml")
16 println "${xmlStart}${value}${xmlEnd}"

```





- 
- 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

```
1 <!-- Endpoint for Google Web APIs -->
2 <service name="GoogleSearchService">
3   <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
4     <soap:address location="http://api.google.com/search/beta2"/>
5   </port>
6 </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?

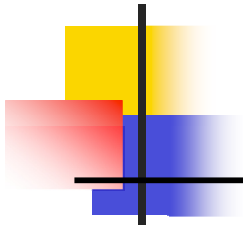
---

- The other white meat:

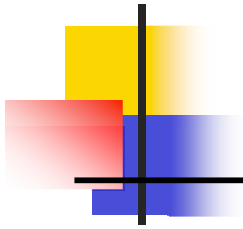
Representational State Transfer (REST) is a software architectural style for distributed hypermedia systems like the world wide web. The term originated in a 2000 doctoral dissertation about the web written by Roy Fielding, one of the principal authors of the HTTP protocol specification, and has quickly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

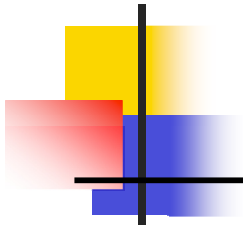
(Source: [http://en.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer))



- 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.

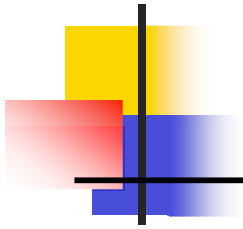
2. Get an Application ID

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:
    - <http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=00000000&query=jboss+at+work>

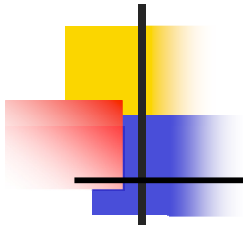




# POX

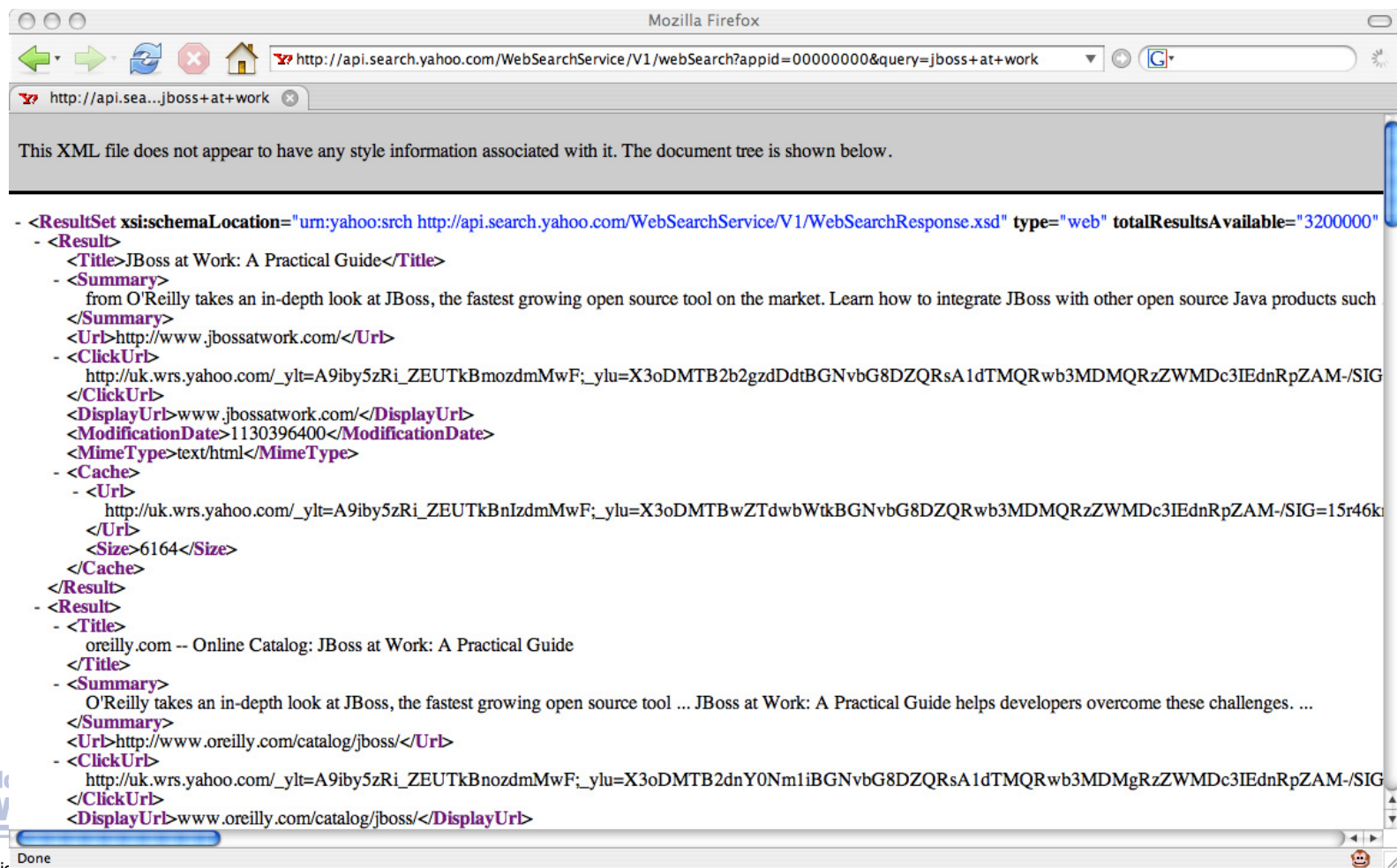
---

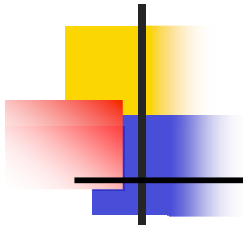
```
- <ResultSet xsi:schemaLocation="urn:yahoo:srch http://api.search.yahoo.com/WebSearchService"
- <Result>
- <Title>
  Amazon.com: JBoss at Work: A Practical Guide: Books
</Title>
- <Summary>
  Amazon.com: JBoss at Work: A Practical Guide: Books by Tom Marrs,Scott Davis ... wh
</Summary>
- <Url>
  http://www.amazon.com/exec/obidos/tg/detail/-/0596007345?v=glance
</Url>
- <ClickUrl>
  http://uk.wrs.yahoo.com/_ylt=A9htfSGYz_pDVWAAATqDdmMwF;_ylu=X3oDMTBjcV
</ClickUrl>
<ModificationDate>1137312000</ModificationDate>
<MimeType>text/html</MimeType>
- <Cache>
- <Url>
  http://uk.wrs.yahoo.com/_ylt=A9htfSGYz_pDVWAAU6DdmMwF;_ylu=X3oDMTA
</Url>
<Size>112159</Size>
</Cache>
</Result>
```



- 
- How do we get valid sample output?
    - All of the tools we examined earlier, plus:
      - The Browser
      - wget

## ■ The Browser





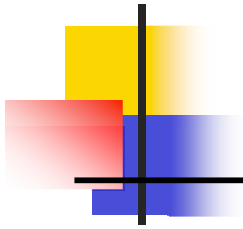
- wget

- <http://www.gnu.org/software/wget/>

- Does basically the same thing as cURL, only the syntax is a bit simpler for GETs

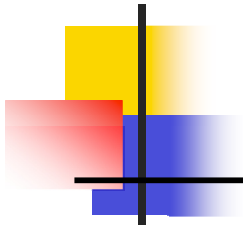


```
wget -O out.xml  
"http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=000000&  
query=jboss+at+work"
```



- Many popular websites offer both SOAP and RESTful interfaces
  - Amazon
    - <http://www.amazon.com/gp/browse.html/102-2243700-0003349?node=3435361>
  - eBay
    - <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 Main

The screenshot shows a web browser window with the title "AWS Zone - Amazon Web Services (TM) - Made Simple. AWS Code Samples, Code Generator REST and SO...". The address bar shows "http://www.awszone.com/". The website has a green header with "AWSzone.com" and two tabs: "Scratch Pads" and "Resources". The main content area is divided into two columns. The left column has a section for "Amazon S3" and a larger section for "Amazon E-Commerce Service". The "Amazon E-Commerce Service" section shows "API Version: 2005-03-23" with links for "WSDL" and "Docs". Below this, it lists "US - United States" and a series of service actions: BrowseNodeLookup, CartAdd, CartCreate, CartClear, CartGet, CartModify, CustomerContentLookup, CustomerContentSearch, Help, ItemLookup, ItemSearch, ListLookup, ListSearch, SellerListingLookup, SellerListingSearch, SellerLookup, SimilarityLookup, and TransactionLookup. The right column is titled "AWS Resources" and contains a list of links: What's New in AWS, AWS Blog, Resource Center, Developer Forums, and Solutions Catalog. The browser's status bar at the bottom shows "Done".

Scratch Pads Resources

Amazon S3

Amazon E-Commerce Service

API Version: 2005-03-23 [WSDL](#) | [Docs](#)

US - United States

- [BrowseNodeLookup](#)
- [CartAdd](#)
- [CartCreate](#)
- [CartClear](#)
- [CartGet](#)
- [CartModify](#)
- [CustomerContentLookup](#)
- [CustomerContentSearch](#)
- [Help](#)
- [ItemLookup](#)
- [ItemSearch](#)
- [ListLookup](#)
- [ListSearch](#)
- [SellerListingLookup](#)
- [SellerListingSearch](#)
- [SellerLookup](#)
- [SimilarityLookup](#)
- [TransactionLookup](#)

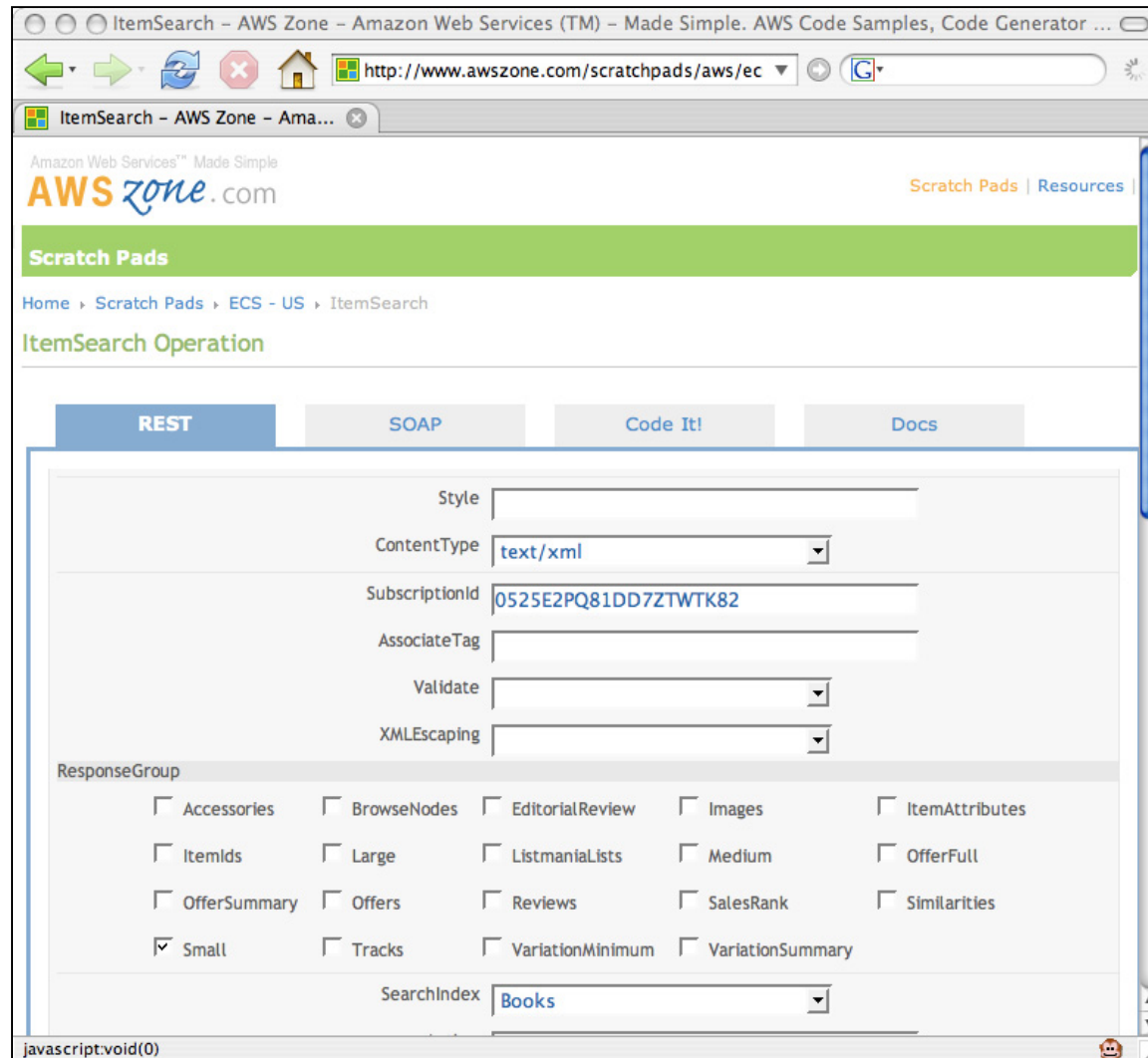
AWS Resources

- [What's New in AWS](#)
- [AWS Blog](#)
- [Resource Center](#)
- [Developer Forums](#)
- [Solutions Catalog](#)

Done



# AWS ItemSearch



The screenshot shows a web browser window with the URL `http://www.awszone.com/scratchpads/aws/ec`. The page title is "ItemSearch - AWS Zone - Amazon Web Services (TM) - Made Simple. AWS Code Samples, Code Generator ...". The page content includes the AWS logo and the text "AWS zone.com". A green bar labeled "Scratch Pads" is visible. Below it, a breadcrumb trail reads "Home > Scratch Pads > ECS - US > ItemSearch". The main heading is "ItemSearch Operation". There are four tabs: "REST" (selected), "SOAP", "Code It!", and "Docs". The "REST" tab contains a form with the following fields:

- Style:
- ContentType:
- SubscriptionId:
- AssociateTag:
- Validate:
- XMLEscaping:

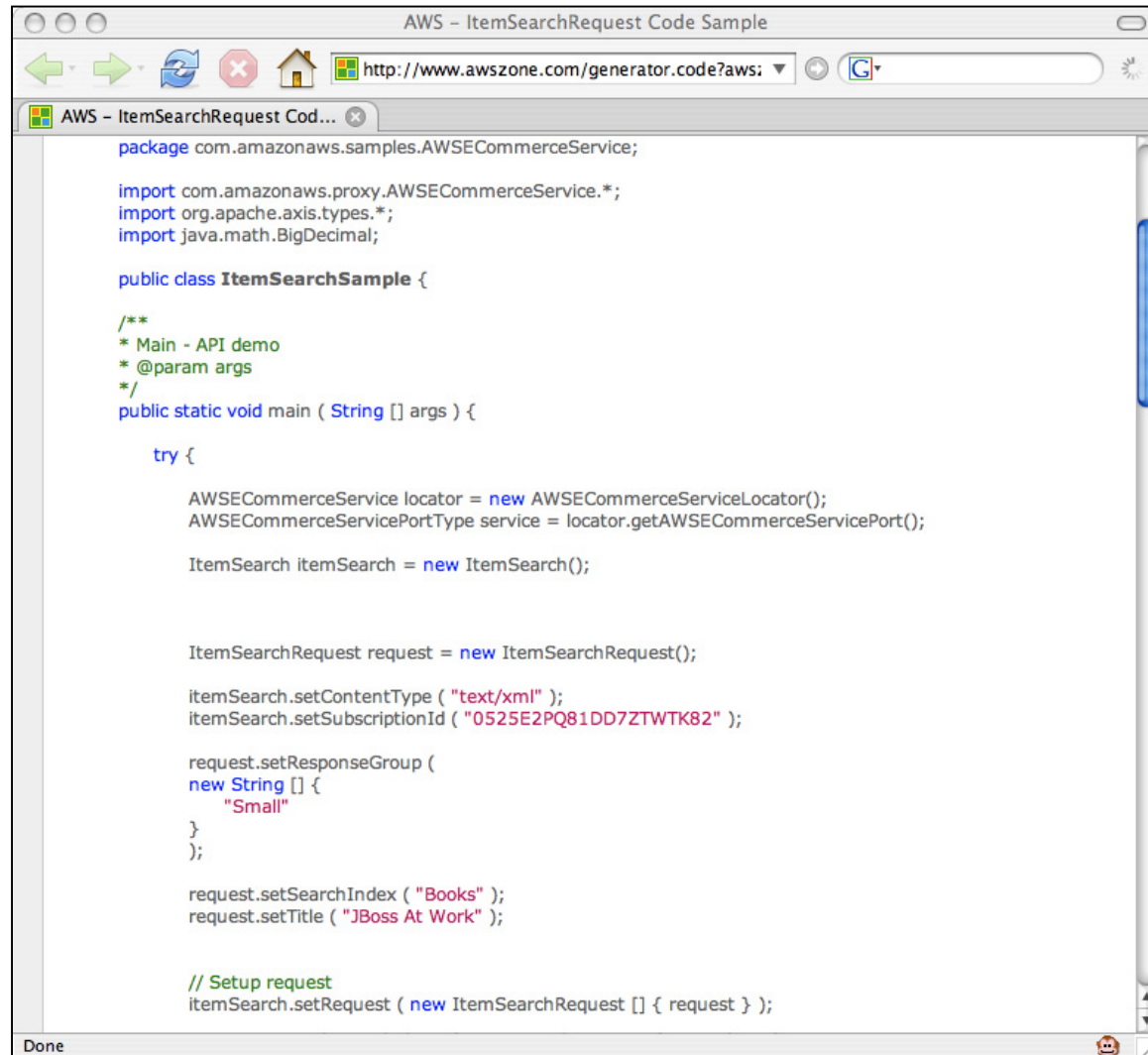
Below these fields is a section titled "ResponseGroup" containing a grid of checkboxes:

|   |                                      |   |   |   |
|---|--------------------------------------|---|---|---|
| <input type="checkbox"/> Accessories      | <input type="checkbox"/> BrowseNodes | <input type="checkbox"/> EditorialReview  | <input type="checkbox"/> Images           | <input type="checkbox"/> ItemAttributes |
| <input type="checkbox"/> ItemIds          | <input type="checkbox"/> Large       | <input type="checkbox"/> ListmaniaLists   | <input type="checkbox"/> Medium           | <input type="checkbox"/> OfferFull      |
| <input type="checkbox"/> OfferSummary     | <input type="checkbox"/> Offers      | <input type="checkbox"/> Reviews          | <input type="checkbox"/> SalesRank        | <input type="checkbox"/> Similarities   |
| <input checked="" type="checkbox"/> Small | <input type="checkbox"/> Tracks      | <input type="checkbox"/> VariationMinimum | <input type="checkbox"/> VariationSummary |   |

At the bottom, there is a "SearchIndex" field with a dropdown menu set to "Books". The status bar at the bottom of the browser window shows "javascript:void(0)".



# AWS Code Generator



```
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();

            itemSearch.setContentType ( "text/xml" );
            itemSearch.setSubscriptionId ( "0525E2PQ81DD7ZTWTk82" );

            request.setResponseGroup (
                new String [] {
                    "Small"
                }
            );

            request.setSearchIndex ( "Books" );
            request.setTitle ( "JBoss 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?

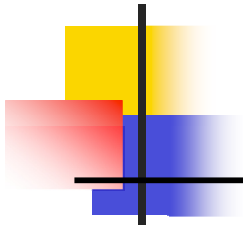
---

- We don't need no stinkin' XML...

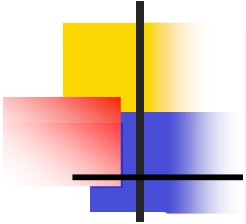
JSON (pronounced Jason), which stands for "JavaScript Object Notation", is a lightweight computer data interchange format. JSON is a subset of the object literal notation of JavaScript but its use does not require JavaScript.

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



## JSON example

[\[edit\]](#)

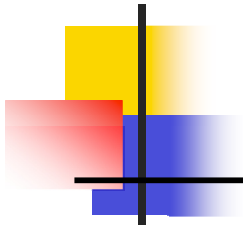
The following is a simple example of a menubar definition using JSON and XML data encodings.

JSON:

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

XML:

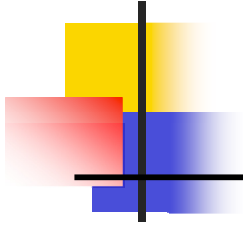
```
<menu id="file" value="File">  
  <popup>  
    <menuitem value="New" onclick="CreateNewDoc()" />  
    <menuitem value="Open" onclick="OpenDoc()" />  
    <menuitem value="Close" onclick="CloseDoc()" />  
  </popup>  
</menu>
```



- 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

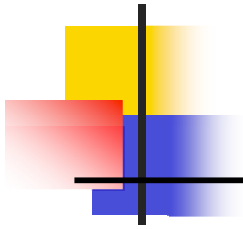
YAML is a recursive acronym meaning "YAML Ain't Markup Language". Early in its development, YAML was said to mean "Yet Another Markup Language", retronymed to distinguish its purpose as data-centric, rather than document markup.

(Source: <http://en.wikipedia.org/wiki/YAML>)



- Even though JSON has “JavaScript” in its name, there are bindings for over 15 other programming languages
  - See <http://www.json.org/> for open source parsers





---

- Getting sample output

➤ wget



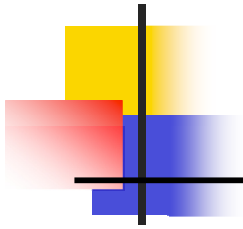
- Yahoo! Supports JSON output by simply adding another query parameter:
  - `output=json`

```
wget -O out.txt "http://api.search.yahoo.com/WebSearchService/V1/webSearch?
appid=000000&
query=jboss+at+work&
output=json"
```

➤ For more information:

- <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](mailto:scottdavis99@yahoo.com)

➤ Download slides:

- <http://www.davisworld.org/presentations>

