

WEBSERVICES WITH WSDL

Sample webservices application and minimalist run-time environment for efficient WSDL interface design.

by Amir Firdus (www.firdus.com)
January 2010

Overview

Designing an interface is not an easy task. Typically, it is an iterative process where each new iteration refines the interface. WSDL adds its own twists. It is not easy to see how the interface definition translates to Java code. Setting up working environment could be a lengthy process. Not to mention that WSDL is hard to fall in love at first sight. This article presents sample webservices application and minimalist run-time environment so that designers can spend more time on the interface design rather than on the tools complexities.

Environment

Environment is as lightweight as can be. Application servers and development tools like Eclipse, JDeveloper or NetBeans are not involved. It consists of:

- JDK 1.6.0_11
- Ant 1.7.1
- JAX-WS 2.1.5

JDK and Ant are prerequisites and need to be installed on the host. Many hosts would have those already installed. In the case that they are not, JDK can be downloaded from Sun (java.sun.com) and Ant from Apache Foundation (www.apache.org). For the convenience, webservices libraries are bundled with the sample application that is accompanying this article. For this reason JAX-WS download is not required but it is highly recommended. Once unzipped sample application has the following directory structure.

Directory	Description
<i>bin</i>	<i>Contains compiled code.</i>
<i>lib</i>	<i>Contains libraries to compile and run the application.</i>
<i>src</i>	<i>Contains application source code.</i>
<i>src-gen</i>	<i>Contains code generated from WSDL.</i>
<i>wSDL</i>	<i>Contains WSDL file.</i>

Complete environment is controlled by Ant targets. They are defined in the

"build.xml" file as follows.

Ant Target	Description
<i>all</i>	<i>Invokes "clean", "generate" and "compile" Ant targets one after another.</i>
<i>clean</i>	<i>Deletes compiled files.</i>
<i>generate</i>	<i>Generates code from WSDL in the src-gen directory.</i>
<i>compile</i>	<i>Compiles both application and code generated from WSDL.</i>
<i>startTestServer</i>	<i>Starts the server where the application is deployed.</i>
<i>runTestClient</i>	<i>Runs test client that makes several webservices calls.</i>

In addition to Ant targets, sample application includes "setenv.bat" file. This script sets Java and Ant variables to the explicitly defined values. ANT_HOME and JAVA_HOME variables should be modified to match Ant and Java installation paths on the host. This script is especially useful in the case when host has multiple Ant and Java installations which, on occasion, can confuse even experienced developers.

```
@echo off
set ANT_HOME=D:\apache-ant-1.7.1
set PATH=%ANT_HOME%\bin;%PATH%
set JAVA_HOME=D:\jdk1.6.0_11
set PATH=%JAVA_HOME%\bin;%PATH%
echo Ant environment set to %ANT_HOME%
echo Java environment set to %JAVA_HOME%
```

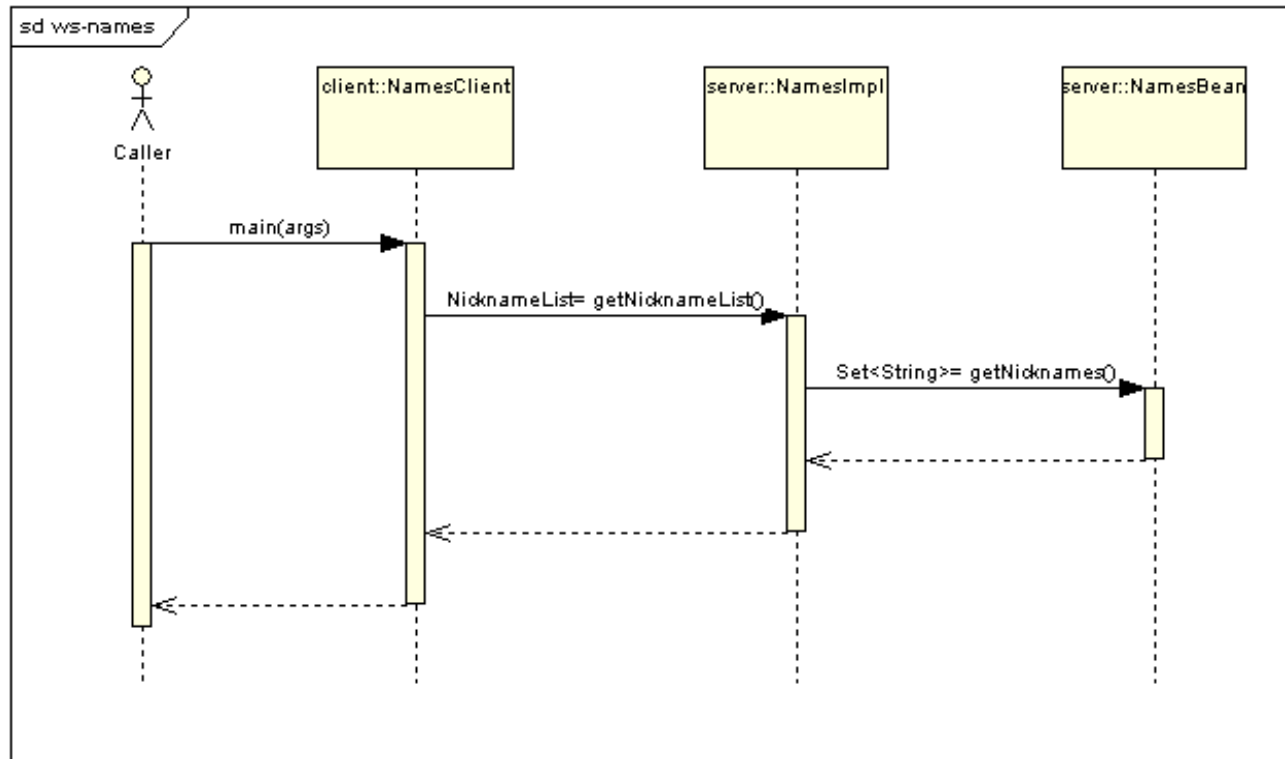
setenv.bat

Sample Application

Sample application is simple. For a given nickname it returns celebrity's full name. Webservices interface has two methods:

- `getNicknameList()`, returns the list of nicknames known to the implementation
- `findByNickname()`, returns full name based on the nickname

The sequence diagram shows how the application works.



Sample Application Sequence Diagram

The application has only five classes. Here is the short description for all of them.

Class	Description
NamesClient.java	Client that makes several webservice calls. Used for runTestClient Ant target.
NamesImpl.java	Webservices implementation. Contains following annotations: @WebService(endpointInterface = "com.xyz.webservices.names.v1_0.Names") @HandlerChain(file = "handlers.xml")
NamesBean.java	Data store and functionality provider. It emulates business tier. Webservices implementation makes calls to this class to get things done.
TestNamesServer.java	Starts webservices server bundled with JDK 1.6
LoggingHandler.java	Logging Handler implementation. It prints XML message to the system console.

Even though the application is simple, WSDL is not.

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://webservices.xyz.com/names/v1_0"
xmlns:tns="http://webservices.xyz.com/names/v1_0"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:obj="http://webservices.xyz.com/schemas/v1_0">

```

```

<types>
  <schema targetNamespace="http://webservices.xyz.com/schemas/v1_0"
    xmlns:tns="http://webservices.xyz.com/schemas/v1_0"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:element name="nickname" type="xsd:string" />
    <xsd:element name="fullname" type="xsd:string" />
    <xsd:complexType name="NicknameList">
      <xsd:sequence>
        <xsd:element name="nicknames" type="xsd:string"
          minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="nicknameList" type="tns:NicknameList" />
  </schema>
</types>
<message name="findByNicknameRequest">
  <part name="nickname" element="obj:nickname" />
</message>
<message name="findByNicknameResponse">
  <part name="fullname" element="obj:fullname" />
</message>
<message name="getNicknameListRequest">
</message>
<message name="getNicknameListResponse">
  <part name="nicknameList" element="obj:nicknameList" />
</message>
<portType name="Names">
  <operation name="findByNickname">
    <input message="tns:findByNicknameRequest" />
    <output message="tns:findByNicknameResponse" />
  </operation>
  <operation name="getNicknameList">
    <input message="tns:getNicknameListRequest" />
    <output message="tns:getNicknameListResponse" />
  </operation>
</portType>
<binding name="NamesBinding" type="tns:Names">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="findByNickname">
    <input><soap:body use="literal" /></input>
    <output><soap:body use="literal" /></output>
  </operation>
  <operation name="getNicknameList">
    <input><soap:body use="literal" /></input>
    <output><soap:body use="literal" /></output>
  </operation>
</binding>
<service name="NamesService">
  <port name="Names" binding="tns:NamesBinding">
    <soap:address

```

```
        location="http://localhost:9191/webservices/names/v1_0" />
    </port>
</service>
</definitions>
```

Sample Application WSDL

As much as the tools have improved, seeing the raw message XML is still the only way to find out what is going on. To do so we have to use handlers. The message is passed from one handler to the next in the handler chain. Each handler has a chance to act on a message. Logging handler prints the message in the XML format to the system console. Handlers are defined in the "handlers.xml" file.

```
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-class>com.xyz.webservices.names.server.LoggingHandler
    </handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

handlers.xml

How To Operate

To operate the environment we need to open two command prompt windows. The first one to build the application and run the client. The second one to run the server from. Both windows have to be opened in the directory where the sample application was unzipped to. For this article that directory is D:\webservices-names.

```
D:\webservices-names>
```

To set Java and Ant variables we run "setenv.bat" script in both command prompt windows.

```
D:\webservices-names>setenv.bat
Ant environment set to D:\apache-ant-1.7.1
Java environment set to D:\jdk1.6.0_11
```

To build the application, we invoke Ant target "all" in the first command prompt window.

```
D:\webservices-names>ant all
Buildfile: build.xml

clean:
generate:
 [wsimport] parsing WSDL...
 [wsimport]
 [wsimport] generating code...
 [wsimport]
```

```
[wsimport] compiling code...
[wsimport]
compile:
[javac] Compiling 5 source files to D:\webservices-names\bin
all:
BUILD SUCCESSFUL
Total time: 9 seconds
```

First Command Prompt Window

To start the server, we invoke Ant target "startTestServer" in the second command prompt window. This command prompt window is now dedicated to running the server. To stop the server we have to hit CTRL+C.

```
D:\webservices-names>ant startTestServer
Buildfile: build.xml

startTestServer:
[java] TestServer started. Hit CTRL+C to stop.
```

Second Command Prompt Window

To make webservice calls we invoke Ant target "runTestClient" from the first command prompt window, the one that is not running the server. If all goes well, client prints the list of nicknames known to the implementation, followed by a series of queries for the celebrity's full name based on the nickname passed as input.

```
D:\webservices-names>ant runTestClient
Buildfile: build.xml

runTestClient:
[java] Names webservices knows following nicknames:
[java] [duke, king]
[java]
[java] What is duke full name?
[java] John Wayne
[java]
[java] What is king full name?
[java] Elvis Presley
[java]
[java] What is snoop dog full name?
[java] Nickname 'snoop dog' not known to Names webservices.

BUILD SUCCESSFUL
Total time: 1 second
```

First Command Prompt Window

The application is configured to print the messages in the XML format on the system console. Because handlers are executed on the server side the messages are printed on the second command prompt window, the one that is running the server. Here is the segment from the text printed there. It shows inbound and

outbound messages for one method call.

```
[java] IN :
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <nickname xmlns="http://webservices.xyz.com/schemas/v1_0">
      duke
    </nickname>
  </S:Body>
</S:Envelope>
[java] OUT:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <fullname xmlns="http://webservices.xyz.com/schemas/v1_0">
      John Wayne
    </fullname>
  </S:Body>
</S:Envelope>
```

Second Command Prompt Window

Conclusion

Interface design is not the goal of this article. Nevertheless, sample application can be a starting point when designing an interface in WSDL. The convenience of having an environment that is lightweight and experimentation friendly makes many tasks so much easier. We can observe how XML namespaces translate to the package names. We can test different versioning strategies necessary for interface life cycle management. We can fine tune WSDL and XSD file organization. And many more ...

References

<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>
http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/
http://blog.vinodsingh.com/2008/09/using-jax-ws-handlers_25.html
<http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>
<http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns3/>

Download

Sample application is available for download at <insert link here>.