WSDL to Java

WSDL to Java

Name

`wsdl2java` - takes a WSDL document and generates fully annotated Java code from which to implement a service.

Synopsis

```
Usage : wsdl2java -fe|-frontend <front-end-name> -db|-databinding <data-binding-name>
-wv <wsdl-version> -p <[wsdl-namespace =]package-name>* -sn <service-name>
-b <binding-file-name>* -reserveClass <class-name>* -catalog <catalog-file-name>
-d <output-directory> -compile -classdir <compile-classes-directory> -impl -server
-client -clientjar <jar-file-name> -all -autoNameResolution -allowElementReferences|-aer<=true>
-defaultValues<=class-name-for-DefaultValueProvider> -ant
-nexclude <schema-namespace [= java-package-name]>* -exsh <(true, false)> -noTypes
-dns <(true, false> -dex <(true, false)> -validate -keep
-wsdlLocation <wsdlLocation> -xjc<xjc-arguments>* -asyncMethods<[=method1,method2,...]>*
-bareMethods<[=method1,method2,...]>* -mimeMethods<[=method1,method2,...]>* -noAddressBinding
-faultSerialVersionUID <fault-serialVersionUID> -exceptionSuper <exceptionSuper>
-mark-generated -suppress-generated-date -maxExtensionStackDepth <maxExtensionStackDepth>
-h|-?|-help -version|-v -verbose|-V -quiet|-q|-Q
-wsdlList <wsdlurl>
```

Description

wsdl2java takes a WSDL document and generates fully annotated Java code from which to implement a service. The WSDL document must have a valid p`ortType` element, but it does not need to contain a binding element or a service element. Using the optional arguments you can customize the generated code. In addition, wsdl2java can generate an Ant based makefile to build your application.

Options

The options used to validate WSDL file are reviewed in the following table.

Option	Interpretation
`-?,-h,-help`	Displays the online help for this utility and exits.
`-fe frontend-name`	Specifies the frontend. Default is JAXWS. Currently supports only JAXWS frontend and a "jaxws21" frontend to generate JAX-WS 2.1 compliant code.
`-db databinding-name`	Specifies the databinding. Default is jaxb. Currently supports jaxb, xmlbeans, sdo (sdo-static and sdo-dynamic), and jibx.
`-wv wsdl-version`	Specifies the wsdl version .Default is WSDL1.1. Currently suppports only WSDL1.1 version.
`-p [ wsdl-namespace= ] PackageName`	Specifies zero, or more, package names to use for the generated code. Optionally specifies the WSDL namespace to package name mapping.
`-sn service-name`	The WSDL service name to use for the generated code.
`-b binding-name`	Specifies JAXWS or JAXB binding files or XMLBeans context files. Use multiple -b flags to specify multiple entries.
`-catalog catalog-file-name`	Specify catalog file to map the imported wsdl/schema
`-d output-directory`	Specifies the directory into which the generated code files are written.
`-compile`	Compiles generated Java files.

`-classdir compile-class-dir`	Specifies the directory into which the compiled class files are written.
`-client`	Generates starting point code for a client mainline.
`-clientjar jar-file-name`	Generates the jar file which contains all the client classes and wsdl;the specified wsdlLocation won't work when the -clientJar is defined.
`-server`	Generates starting point code for a server mainline.
`-impl`	Generates starting point code for an implementation object.
`-all`	Generates all starting point code: types, service proxy, service interface, server mainline, client mainline, implementation object, and an Ant `build.xml` file.
`-ant`	Specify to generate an Ant `build.xml` script.
`-autoNameResolution`	Automatically resolve naming conflicts without requiring the use of binding customizations.
`-defaultValues=[DefaultValueProvider impl]`	Specifies that default values are generated for the impl and client. You can also provide a custom default value provider. The default provider is `RandomValueProvider`
`-nexclude schema-namespace [=java-packagename]`	Ignore the specified WSDL schema namespace when generating code. This option may be specified multiple times. Also, optionally specifies the Java package name used by types described in the excluded namespace(s).
`-exsh (true /false)`	Enables or disables processing of implicit SOAP headers (i.e. SOAP headers defined in the wsdl:binding but not wsdl:portType section.) Processing the SOAP headers requires the SOAP binding jars available on the classpath which was not the default in CXF 2.4.x and older. You may need to add a dependency to cxf-rt-binding-soap for this flag to work. Default is false.
`-dns (true /false)`	Enables or disables the loading of the default namespace package name mapping. Default is true and http://www.w3.org/2005/08/addressing=org.apache.cxf.ws.addressing namespace package mapping will be enabled.
`-dex (true /false)`	Enables or disables the loading of the default excludes namespace mapping. Default is true.
`-validate`	Enables validating the WSDL before generating the code.
`-keep`	Specifies that the code generator will not overwrite any preexisting files. You will be responsible for resolving any resulting compilation issues.
`-wsdlLocation wsdlLocation`	Specifies the value of the @WebServiceClient annotation's wsdlLocation property.
`-xjc<xjc args>`	Specifies a comma separated list of arguments that are passed directly to the XJC processor when using the JAXB databinding. A list of available XJC plugins can be obtained using `-xjc-X`.
`-noAddressBinding`	For compatibility with CXF 2.0, this flag directs the code generator to generate the older CXF proprietary WS-Addressing types instead of the JAX-WS 2.1 compliant WS-Addressing types.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.
`-exceptionSuper`	superclass for fault beans generated from wsdl:fault elements (defaults to java.lang.Exception)
`-reserveClass classname`	Used with -autoNameResolution, defines a class names for wsdl-to-java **not** to use when generating classes. Use this option multiple times for multiple classes.
`-allowElementReferences<=true>`	(or -aer) If true, disregards the rule given in section 2.3.1.2(v) of the JAX-WS 2.2 specification disallowing element references when using wrapper-style mapping.
`-asyncMethods=foo,bar,...`	List of subsequently generated Java class methods to allow for client-side asynchronous calls, similar to enableAsyncMapping in a JAX-WS binding file.
`-bareMethods=foo,bar,...`	List of subsequently generated Java class methods to have wrapper style (see below), similar to enableWrapperStyle in JAX-WS binding file.

`-mimeMethods=foo,bar,...`	List of subsequently generated Java class methods to enable mime:content mapping, similar to enableMIMEContent in JAX-WS binding file.
`-faultSerialVersionUID <fault-serialVersionUID>`	How to generate suid of fault exceptions. Use NONE, TIMESTAMP, FQCN, or a specific number. Default is NONE.
`-mark-generated`	Adds the @Generated annotation to classes generated.
`-suppress-generated-date`	Suppresses writing the current timestamp in the generated file (since CXF version 3.2.2)
-maxExtensionStackDepth <int>	The maximum stack depth allowed when parsing schema extensions. The default is 5. (since **CXF 3.3.4**)
wsdlurl	The path and name of the WSDL file to use in generating the code.

You must specify the absolute or relative path to the WSDL document as the last argument.

Examples

```
wsdl2java HelloWorld.wsdl

wsdl2java -p com.mycompany.greeting Greeting.wsdl

wsdl2java -client HelloWorld.wsdl
```

(See below for usage with Apache Ant and Maven.)

Using wsdl2java with Ant

The wsdl2java command can be wrapped inside an Ant target as shown below:

```
<?xml version="1.0"?>
<project name="cxf wsdl2java" basedir=".">
   <property name="cxf.home" location ="/usr/myapps/cxf-2.5.1"/>

   <path id="cxf.classpath">
      <fileset dir="${cxf.home}/lib">
         <include name="*.jar"/>
      </fileset>
   </path>

   <target name="cxfWSDLToJava">
      <java classname="org.apache.cxf.tools.wsdlto.WSDLToJava" fork="true">
         <arg value="-client"/>
         <arg value="-d"/>
         <arg value="src"/>
         <arg value="MyWSDL.wsdl"/>
         <classpath>
            <path refid="cxf.classpath"/>
         </classpath>
      </java>
   </target>
</project>
```

Make sure you set the "fork=true" attribute for the <java/> task as shown above. Also, remember to keep each word or flag within the command line options in its own <arg/> element (e.g., do not use <arg value="-d src"/>, but split them up into two <arg/> elements as done here.)

Although we would recommend using Maven, see the antbuild sample in the CXF distribution for an example of using Ant to create a CXF project.

JAXWS Customization

The default JAX-WS frontend created by wsdl2java can be customized by using a customization binding file. For an example, see the async_binding.xml file in samples/jaxws_async – if specified when running wsdl2java, it will generate asynchronous methods in the SEI.

Q: What if I want to change the generated SEI name?

A: We don't have a command-line option to do this, but you can have a binding file like the following snippet to achieve this goal

```
<bindings
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    wsdlLocation="hello_world.wsdl"
    xmlns="http://java.sun.com/xml/ns/jaxws">
    <bindings node="wsdl:definitions/wsdl:portType[@name='GreeterSE']">
        <class name="GreeterSEI"/>
    </bindings>
</bindings>
```

Q: How do I pass the binding file to wsdl2java?

A: If you are using wsdl2java via command line tool:

wsdl2java HelloWorld.wsdl -b my_binding.xml

For Ant, follow the example above on how to add "-b" and "my_binding.xml" as arg elements.

For Maven see cxf-codegen-plugin

Q: How to map xsd:dateTime to java.util.Date?
Well, people don't like the XMLGregorianCalendar which is the default mapping from the xsd:dateTime (xsd:time and xsd:date as well),
We can use the jaxws customization to change the default mapping, here are some sample binding files
If you have schema inside the wsdl, here is the binding file you can use:

```
<jaxws:bindings wsdlLocation="YOUR_WSDL_LOCATION"
          xmlns:jaxws="http://java.sun.com/xml/ns/jaxws"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
          xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <jaxws:bindings  node="wsdl:definitions/wsdl:types/xs:schema[@targetNamespace='THE_NAMESPACE_OF_YOUR_SCHEMA']"
>
      <jxb:globalBindings xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:xs="http://www.w3.org/2001
/XMLSchema">
        <jxb:javaType name="java.util.Date" xmlType="xs:dateTime"
                      parseMethod="org.apache.cxf.xjc.runtime.DataTypeAdapter.parseDateTime"
                      printMethod="org.apache.cxf.xjc.runtime.DataTypeAdapter.printDateTime"/>
      </jxb:globalBindings>
  </jaxws:bindings>
</jaxws:bindings>
```

This requires an additional dependency:

```
<dependency>
   <groupId>org.apache.cxf.xjc-utils</groupId>
   <artifactId>cxf-xjc-runtime</artifactId>
</dependency>
```

If you want to use java.util.Calendar, just change the org.apache.cxf.xjc.runtime.DataTypeAdapter to javax.xml.bind.DatatypeConverter, and change the name value to "java.util.Calendar"

If your schema is out of wsdl, here is an example you can try:

```
<jxb:bindings version="2.0"
              xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
              xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <jxb:bindings schemaLocation="file:<path><name>.xsd" node="/xs:schema">
    <jxb:globalBindings xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <jxb:javaType name="java.util.Date" xmlType="xs:dateTime"
                    parseMethod="org.apache.cxf.xjc.runtime.DataTypeAdapter.parseDateTime"
                    printMethod="org.apache.cxf.xjc.runtime.DataTypeAdapter.printDateTime"/>
    </jxb:globalBindings>
  </jxb:bindings>
</jxb:bindings>
```

Q: How can I switch my generated web service method calls from wrapper style to non wrapper-style (or vice-versa)?

A: Create an external binding file and set the value of <enableWrapperStyle/> to true or false as desired:

```
<jaxws:bindings
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    wsdlLocation="your.wsdl"
    xmlns="http://java.sun.com/xml/ns/jaxws"
    xmlns:jaxws="http://java.sun.com/xml/ns/jaxws">
    <enableWrapperStyle>false</enableWrapperStyle>
</jaxws:bindings>
```

Alternatively you can embed this instruction within the WSDL file directly, as the immediate child of the wsdl:portType:

```
<wsdl:portType name="MyWebServicePortType">
    <jaxws:bindings xmlns:jaxws="http://java.sun.com/xml/ns/jaxws">
        <jaxws:enableWrapperStyle>false</jaxws:enableWrapperStyle>
        ... other binding settings if needed ...
    </jaxws:bindings>
    <wsdl:operation name="sayHello">
    ...
</wsdl:portType>
```

Note: The meaning of "wrapper-style" and "non-wrapper style" as defined in the JAX-WS 2.1 specification can be counterintuitive. Wrapper-style indicates that each data element within the request message gets its own Java parameter, while non-wrapper style means that a single Java object containing all the data elements serves as the lone parameter to the web service method call. (See Figure 2.2 of the specification for an example.) Also, **note the wrapper style is not always available**, the WSDL criteria specified in Section 2.3.1.2 ("Wrapper Style") of the specification must be met or only non-wrapper style will be generated.

Q: What else can I change with the JAXWS customization binding file?

A: You can find the full list of customization items in Chapter 8 of the JAX-WS Specification.

Using maven to generate java code from WSDL

see cxf-codegen-plugin

See Also

`idl2wsdl`, `java2js`, `java2ws`, `wsdl2corba`, `wsdl2js`, `wsdl2service`, `wsdl2soap`, `wsdl2xml`, `wsdlvalidator` and xsd2wsdl.