WebServices - Axis

1. Axis Reference Guide

1.2 Version
Feedback: axis-dev@ws.apache.org

1.1. Table of Contents

1.2. Tools Reference

1.2.1. WSDL2Java Reference

Usage: java org.apache.axis.wsdl.WSDL2Java [options] WSDL-URI
Options:

```
-h, --help
  print this message and exit
-v, --verbose
  print informational messages
-n, --noImports
  only generate code for the immediate WSDL document
-O, --timeout <argument>
  timeout in seconds (default is 45, specify -1 to disable)
-D, --Debug
  print debug information
-W, --noWrapped
  turn off support for "wrapped" document/literal
-s, --server-side
```

```
  emit server-side bindings for web service
-S, --skeletonDeploy <argument>
  deploy skeleton (true) or implementation (false) in
  deploy.wsdd.
  Default is false.  Assumes --server-side.
-N, --NStoPkg <argument>=<value>
  mapping of namespace to package
-f, --fileNStoPkg <argument>
  file of NStoPkg mappings (default NStoPkg.properties)
-p, --package <argument>
  override all namespace to package mappings, use this package
  name instead
-o, --output <argument>
  output directory for emitted files
-d, --deployScope <argument>
  add scope to deploy.xml: "Application", "Request", "Session"
-t, --testCase
  emit junit testcase class for web service
-a, --all
  generate code for all elements, even unreferenced ones
-T, --typeMappingVersion
  indicate 1.1 or 1.2. The default is 1.1 (SOAP 1.1 JAX-RPC  compliant.
  1.2 indicates SOAP 1.1 encoded.)
-F, --factory <argument>
  name of a custom class that implements GeneratorFactory interface
  (for extending Java generation functions)
-i, --nsInclude <namespace>
  namescape to specifically include in the generated code (defaults to
  all namespaces unless specifically excluded with the -x option)
-x, --nsExclude <namespace>
  namespace to specifically exclude from the generated code (defaults to
  none excluded until first namespace included with -i option)
-p, --property <name>=<value>
  name and value of a property for use by the custom GeneratorFactory
-H, --helperGen
  emits separate Helper classes for meta data
-U, --user <argument>
  username to access the WSDL-URI
-P, --password <argument>
  password to access the WSDL-URI
-c, --implementationClassName <argument>
  use this as the implementation class
```

-h, --help

Print the usage statement and exit

-v, --verbose

See what the tool is generating as it is generating it.

-n, --noImports

Only generate code for the WSDL document that appears on the command line. The default behaviour is to generate files for all WSDL documents, the immediate one and all imported ones.

-O, --timeout

Timeout in seconds. The default is 45. Use -1 to disable the timeout.

-D, --Debug

Print debug information, which currently is WSDL2Java's symbol table. Note that this is only printed after the symbol table is complete, ie., after the WSDL is parsed successfully.

-W, --noWrapped

This turns off the special treatment of what is called "wrapped" document/literal style operations. By default, WSDL2Java will recognize the following conditions:

- If an input message has is a single part.
- The part is an element.
- The element has the same name as the operation
- The element's complex type has no attributes

When it sees this, WSDL2Java will 'unwrap' the top level element, and treat each of the components of the element as arguments to the operation. This type of WSDL is the default for Microsoft .NET web services, which wrap up RPC style arguments in this top level schema element.

-s, --server-side

Emit the server-side bindings for the web service:

- a skeleton class named <bindingName>Skeleton. This may or may not be emitted (see -S, --skeletonDeploy).
- an implementation template class named <bindingName>Impl. Note that, if this class already exists, then it is not emitted.
- deploy.wsdd
- undeploy.wsdd

-S, --skeletonDeploy <argument>

Deploy either the skeleton (true) or the implementation (false) in deploy.wsdd. In other words, for "true" the service clause in the deploy.wsdd file will look something like:

```
<service name="AddressBook" provider="java:RPC">
 <parameter name="className" value="samples.addr.AddressBookSOAPBindingSkeleton"/>
 ...
</service>
```

and for "false" it would look like:

```
<service name="AddressBook" provider="java:RPC">
 <parameter name="className" value="samples.addr.AddressBookSOAPBindingImpl"/>
   ...
</service>
```

The default for this option is false. When you use this option, the --server-side option is assumed, so you don't have to explicitly specify --server-side as well.

-N, --NStoPkg <argument>=<value>

By default, package names are generated from the namespace strings in the WSDL document in a magical manner (typically, if the namespace is of the form

"http://x.y.com" or "urn:x.y.com" the corresponding package will be "com.y.x"). If this magic is not what you want, you can provide your own mapping using the --NStoPkg argument, which can be repeated as often as necessary, once for each unique namespace mapping. For example, if there is a namespace in the WSDL document called "urn:AddressFetcher2", and you want files generated from the objects within this namespace to reside in the package samples.addr, you would provide the following option to WSDL2Java:

```
--NStoPkg urn:AddressFetcher2=samples.addr
```

(Note that if you use the short option tag, "-N", then there must not be a space between "-N" and the namespace.)

-f, --fileNStoPkg <argument>

If there are a number of namespaces in the WSDL document, listing a mapping for them all could become tedious. To help keep the command line terse, WSDL2Java will also look for mappings in a properties file. By default, this file is named "NStoPkg.properties" and it must reside in the default package (ie., no package). But you can explicitly provide your own file using the --fileNStoPkg option.

The entries in this file are of the same form as the arguments to the --NStoPkg command line option. For example, instead of providing the command line option as above, we could provide the same information in NStoPkg.properties:

```
urn\:AddressFetcher2=samples.addr
```

(Note that the colon must be escaped in the properties file.)

If an entry for a given mapping exists both on the command line and in the properties file, the command line entry takes precedence.

-p, --package <argument>

This is a shorthand option to map all namespaces in a WSDL document to the same Java package name. This can be useful, but dangerous. You must make sure that you understand the effects of doing this. For instance there may be multiple types with the same name in different namespaces. It is an error to use the --NStoPkg switch and --package at the same time.

-o, --output <argument>

The root directory for all emitted files.

-d, --deployScope <argument>

Add scope to deploy.wsdd: "Application", "Request", or "Session". If this option does not appear, no scope tag appears in deploy.wsdd, which the Axis runtime defaults to "Request".

-t, --testCase

Generate a client-side JUnit test case. This test case can stand on its own, but it doesn't really do anything except pass default values (null for objects, 0 or false for primitive types). Like the generated implementation file, the generated test

case file could be considered a template that you may fill in.

-a, --all

Generate code for all elements, even unreferenced ones. By default, WSDL2Java only generates code for those elements in the WSDL file that are referenced.

A note about what it means to be referenced. We cannot simply say: start with the services, generate all bindings referenced by the service, generate all portTypes referenced by the referenced bindings, etc. What if we're generating code from a WSDL file that only contains portTypes, messages, and types? If WSDL2Java used service as an anchor, and there's no service in the file, then nothing will be generated. So the anchor is the lowest element that exists in the WSDL file in the order:

1. types
2. portTypes
3. bindings
4. services

For example, if a WSDL file only contained types, then all the listed types would be generated. But if a WSDL file contained types and a portType, then that portType will be generated and only those types that are referenced by that portType.

Note that the anchor is searched for in the WSDL file appearing on the command line, not in imported WSDL files. This allows one WSDL file to import constructs defined in another WSDL file without the nuisance of having all the imported WSDL file's constructs generated.

-T, --typeMappingVersion <argument>

Indicate 1.1 or 1.2. The default is 1.2 (SOAP 1.2 JAX-RPC compliant).

-F, --factory <argument>

Used to extend the functionality of the WSDL2Java emitter. The argument is the name of a class which extends JavaWriterFactory.

-H, --helperGen

Emits separate Helper classes for meta data.

-U, --user <argument>

This username is used in resolving the WSDL-URI provided as the input to WSDL2Java. If the URI contains a username, this will override the command line switch. An example of a URL with a username and password is:

```
http://user:password@hostname:port/path/to/service?WSDL
```

-P, --password <argument>

This password is used in resolving the WSDL-URI provided as the input to WSDL2Java. If the URI contains a password, this will override the command line switch.

-c, --implementationClassName <argument>
Set the name of the implementation class.Especially useful when exporting an existing class as a web service using java2wsdl followed by wsdl2java. If you are using the skeleton deploy option you must make sure, after generation, that your implementation class implements the port type name interface generated by wsdl2java. You should also make sure that all your exported methods throws java.lang.RemoteException.

1.2.2. Java2WSDL Reference

Here is the help message generated from the current tool:

Java2WSDL emitter
Usage: java org.apache.axis.wsdl.Java2WSDL [options] class-of-portType
Options:

```
-h, --help
  print this message and exit
-I, --input <argument>
  input WSDL filename
-o, --output <argument>
  output WSDL filename
-l, --location <argument>
  service location url
-P, --portTypeName <argument>
  portType name (obtained from class-of-portType if not specified)
-b, --bindingName <argument>
  binding name (--servicePortName value + "SOAPBinding" if not specified)
-S, --serviceElementName <argument>
  service element name (defaults to --servicePortName value + "Service")
-s, --servicePortName <argument>
  service port name (obtained from --location if not specified)
-n, --namespace <argument>
  target namespace
-p, --PkgtoNS <argument>=<value>
  package=namespace, name value pairs
-m, --methods <argument>
  space or comma separated list of methods to export
-a, --all
  look for allowed methods in inherited class
-w, --outputWsdlMode <argument>
  output WSDL mode: All, Interface, Implementation
-L, --locationImport <argument>
  location of interface wsdl
-N, --namespaceImpl <argument>
  target namespace for implementation wsdl
-O, --outputImpl <argument>
  output Implementation WSDL filename, setting this causes
  --outputWsdlMode to be ignored
-i, --implClass <argument>
  optional class that contains implementation of methods in class-of-portType.
```

```
  The debug information in the class is used
  to obtain the method parameter names, which are used to set
  the WSDL part names.
-x, --exclude <argument>
  space or comma separated list of methods not to export
-c, --stopClasses <argument>
  space or comma separated list of class names which will stop
  inheritance search if --all switch is given
-T, --typeMappingVersion <argument>
  indicate 1.1 or 1.2.  The default is 1.1 (SOAP 1.1 JAX-RPC
  compliant  1.2 indicates SOAP 1.1 encoded.)
-A, --soapAction <argument>
  value of the operations soapAction field. Values are DEFAULT,
  OPERATION or NONE. OPERATION forces soapAction to the name
  of the operation.  DEFAULT causes the soapAction to be set
  according to the operations meta data (usually "").  NONE forces
  the soapAction to "".  The default is DEFAULT.
-y, --style <argument>
  The style of binding in the WSDL, either DOCUMENT, RPC, or WRAPPED.
-u, --use <argument>
  The use of items in the binding, either LITERAL or ENCODED
-e, --extraClasses <argument>
  A space or comma separated list of class names to be added to
  the type section.
-C, --importSchema
  A file or URL to an XML Schema that should be physically
  imported into the generated WSDL
-X, --classpath
  additional classpath elements
Details:
   portType element name= <--portTypeName value> OR <class-of-portType name>
   binding  element name= <--bindingName value> OR <--servicePortName value>
Soap Binding
   service  element name= <--serviceElementName value> OR <--portTypeName value>
Service
   port  element name= <--servicePortName value>
   address location  = <--location value>
```

-h , --help

Prints the help message.

-I, --input <WSDL file>

Optional parameter that indicates the name of the input wsdl file. The output wsdl file will contain everything from the input wsdl file plus the new constructs. If a new construct is already present in the input wsdl file, it is not added. This option is useful for constructing a wsdl file with multiple ports, bindings, or portTypes.

-o, --output <WSDL file>

Indicates the name of the output WSDL file. If not specified, a suitable default WSDL file is written into the current directory.

-l, --location <location>

Indicates the url of the location of the service. The name after the last slash or

backslash is the name of the service port (unless overridden by the -s option). The service port address location attribute is assigned the specified value.

-P, --portTypeName <name>
Indicates the name to use for the portType element. If not specified, the class-of-portType name is used.

-b, --bindingName <name>
Indicates the name to use for the binding element. If not specified, the value of the --servicePortName + "SoapBinding" is used.

-S, --serviceElementName <name>
Indicates the name of the service element. If not specified, the service element is the <portTypeName>Service.

-s, --servicePortName <name>
Indicates the name of the service port. If not specified, the service port name is derived from the --location value.

-n, --namespace <target namespace>
Indicates the name of the target namespace of the WSDL.

-p, --PkgToNS <package> <namespace>
Indicates the mapping of a package to a namespace. If a package is encountered that does not have a namespace, the Java2WSDL emitter will generate a suitable namespace name. This option may be specified multiple times.

-m, --methods <arguments>
If this option is specified, only the indicated methods in your interface class will be exported into the WSDL file. The methods list must be comma separated. If not specified, all methods declared in the interface class will be exported into the WSDL file.

-a, --all
If this option is specified, the Java2WSDL parser will look into extended classes to determine the list of methods to export into the WSDL file.

-w, --outputWSDLMode <mode>
Indicates the kind of WSDL to generate. Accepted values are:

- All --- (default) Generates wsdl containing both interface and implementation WSDL constructs.
- Interface --- Generates a WSDL containing the interface constructs (no service element).
- Implementation -- Generates a WSDL containing the implementation. The interface WSDL is imported via the -L option.

-L, --locationImport <url>
Used to indicate the location of the interface WSDL when generating an implementation WSDL.

-N, --namespaceImpl <namespace>
Namespace of the implementation WSDL.
-O, --outputImpl <WSDL file>
Use this option to indicate the name of the output implementation WSDL file. If specified, Java2WSDL will produce interface and implementation WSDL files. If this option is used, the -w option is ignored.
-i, --implClass <impl-class>
Sometimes extra information is available in the implementation class file. Use this option to specify the implementation class.
-x, --exclude <list>
List of methods to not exclude from the wsdl file.
-c, --stopClasses <list>
List of classes which stop the Java2WSDL inheritance search.
-T, --typeMappingVersion <version>
Choose the default type mapping registry to use. Either 1.1 or 1.2.
-A, --soapAction <argument>
The value of the operations soapAction field. Values are DEFAULT, OPERATION or NONE. OPERATION forces soapAction to the name of the operation. DEFAULT causes the soapAction to be set according to the operation's meta data (usually ""). NONE forces the soapAction to "". The default is DEFAULT.
-y, --style <argument>
The style of the WSDL document: RPC, DOCUMENT or WRAPPED. The default is RPC. If RPC is specified, an rpc wsdl is generated. If DOCUMENT is specified, a document wsdl is generated. If WRAPPED is specified, a document/literal wsdl is generated using the wrapped approach. Wrapped style forces the use attribute to be literal.
-u, --use <argument>
The use of the WSDL document: LITERAL or ENCODED. If LITERAL is specified, the XML Schema defines the representation of the XML for the request. If ENCODED is specified, SOAP encoding is specified in the generated WSDL.
-e, --extraClasses <argument>
Specify a space or comma seperated list of class names which should be included in the types section of the WSDL document. This is useful in the case where your service interface references a base class and you would like your WSDL to contain XML Schema type defintions for these other classes. The --extraClasses option can be specified duplicate times. Each specification results in the additional classes being added to the list.

-C, --importSchema
A file or URL to an XML Schema that should be physically imported into the generated WSDL
-X, --classpath
Additional classpath elements

1.3. Deployment (WSDD) Reference

Note : all the elements referred to in this section are in the WSDD namespace, namely "http://xml.apache.org/axis/wsdd/".

<deployment>
The root element of the deployment document which tells the Axis engine that this is a deployment. A deployment document may represent EITHER a complete engine configuration OR a set of components to deploy into an active engine.
<GlobalConfiguration>
This element is used to control the engine-wide configuration of Axis. It may contain several subelements:

- <parameter> : This is used to set options on the Axis engine - see the Global Axis Configuration section below for more details. Any number of <parameter> elements may appear.
- <role> : This is used to set a SOAP actor/role URI which the engine will recognize. This allows SOAP headers targeted at that role to be successfully processed by the engine. Any number of <role> elements may appear.
- <requestFlow> : This is used to configure global request Handlers, which will be invoked before the actual service on every request. You may put any number of <handler> or <chain> elements (see below) inside the <requestFlow>, but there may only be one <requestFlow>.
- <responseFlow> : This is used to configure global response Handlers, which will be invoked after the actual service on every request. You may put any number of <handler> or <chain> elements (see below) inside the <responseFlow>, but there may only be one <responseFlow>.

<undeployment>
The root element of the deployment document which tells Axis that this is an undeployment.
<handler [name="name"] type="type"/>
Belongs at the top level inside a <deployment> or <undeployment>, or inside a <chain>, <requestFlow>, or <responseFlow>. Defines a Handler, and indicates the type of the handler. "Type" is either the name of another previously defined Handler, or a QName of the form "java:class.name". The optional "name" attribute allows you to refer to this Handler definition in other parts of the

deployment. May contain an arbitrary number of <parameter name="name" value="value"> elements, each of which will supply an option to the deployed Handler.

<service name="name" provider="provider" >

Deploys/undeploys an Axis Service. This is the most complex WSDD tag, so we're going to spend a little time on it.

Options may be specified as follows : <parameter name="name" value="value"/>, and common ones include:

- className : the backend implementation class
- allowedMethods : Each provider can determine which methods are allowed to be exposed as web services.

  To summaries for Axis supplied providers:

  Java RPC Provider (provider="java:RPC") by default all public methods specified by the class in the className option, including any inherited methods are available as web services.

  For more details regarding the Java Provider please see WHERE???.

  Java MsgProvder (provider="java:MSG")

  In order to further restrict the above methods, the allowedMethods option may be used to specify in a space delimited list the names of only those methods which are allowed as web services. It is also possible to specify for this option the value "*" which is functionally equivalent to not specify the option at all. Also, it is worth mentioning that the operation element is used to further define the methods being offered, but it does not affect which methods are made available.

  Note, while this is true for Axis supplied providers, it is implementation dependent on each individual provider. Please review your providers documentation on how or if it supports this option.

  Note, Exposing any web service has security implications.
  As a best practices guide it is highly recommend when offering a web service in unsecure environment to restrict allowed methods to only those required for the service being offered. And, for those that are made available, to fully understand their function and how they may access and expose your systems's resources.
- allowedRoles : comma-separated list of roles allowed to access this service (Note that these are security roles, as opposed to SOAP roles. Security roles control access, SOAP roles control which SOAP headers are processed.)
- extraClasses : Specify a space or comma seperated list of class names which should be included in the types section of the WSDL document. This is useful in the case where your service interface references a base class and you would like your WSDL

to contain XML Schema type defintions for these other classes.

If you wish to define handlers which should be invoked either before or after the service's provider, you may do so with the <requestFlow> and the <responseFlow> subelements. Either of those elements may be specified inside the <service> element, and their semantics are identical to the <chain> element described below - in other words, they may contain <handler> and <chain> elements which will be invoked in the order they are specified.

To control the roles that should be recognized by your service Handlers, you can specify any number of <role> elements inside the service declaration.

Example:

```
<service name="test">
  <parameter name="className" value="test.Implementation"/>
  <parameter name="allowedMethods" value="*"/>
  <namespace>http://testservice/</namespace>
  <role>http://testservice/MyRole</role>
  <requestFlow> <!-- Run these before processing the request -->
    <handler type="java:MyHandlerClass"/>
    <handler type="somethingIDefinedPreviously"/>
  </requestFlow>
</service>
```

Metadata may be specified about particular operations in your service by using the <operation> tag inside a service. This enables you to map the java parameter names of a method to particular XML names, to specify the parameter modes for your parameters, and to map particular XML names to particular operations.

```
<operation name="method">
</operation>
```

<chain name="name"> <subelement/>... </chain>

Defines a chain. Each handler (i.e. deployed handler name) in the list will be invoked() in turn when the chain is invoked. This enables you to build up "modules" of commonly used functionality. The subelements inside chains may be <handler>s or <chain>s. <handler>s inside a <chain> may either be defined in terms of their Java class:

```
<chain name="myChain">
  <handler type="java:org.apache.axis.handlers.LogHandler"/>
</chain>
```

or may refer to previously defined <handlers>, with the "type" of the handler referring to the name of the other handler definition:

```
<handler name="logger" type="java:org.apache.axis.handlers.LogHandler"/>
<chain name="myChain"/>
   <handler type="logger"/>
</chain>
```

<transport name="name">

Defines a transport on the server side. Server transports are invoked when an incoming request arrives. A server transport may define <requestFlow> and/or <responseFlow> elements to specify handlers/chains which should be invoked

during the request (i.e. incoming message) or response (i.e. outgoing message) portion of processing (this function works just like the <service> element above). Typically handlers in the transport request/response flows implement transport-specific functionality, such as parsing protocol headers, etc.
For any kind of transport (though usually this relates to HTTP transports), users may allow Axis servlets to perform arbitrary actions (by means of a "plug-in") when specific query strings are passed to the servlet (see the section Axis Servlet Query String Plug-ins in the Axis Developer's Guide for more information on what this means and how to create a plug-in). When the name of a query string handler class is known, users can enable it by adding an appropriate <parameter> element in the Axis server configuration's <transport> element. An example configuration might look like the following:

```
<transport name="http">
  <parameter name="useDefaultQueryStrings" value="false" />
  <parameter name="qs.name" value="class.name" />
</transport>
```

In this example, the query string that the Axis servlet should respond to is ?name and the class that it should invoke when this query string is encountered is named `class.name`. The `name` attribute of the <parameter> element must start with the string "qs." to indicate that this <parameter> element defines a query string handler. The `value` attribute must point to the name of a class implementing the `org.apache.axis.transport.http.QSHandler` interface. By default, Axis provides for three Axis servlet query string handlers (?list, ?method, and ?wsdl). See the Axis server configuration file for their definitions. If the user wishes not to use these default query string handlers (as in the example), a <parameter> element with a `name` attribute equal to "useDefaultQueryStrings" should have its `value` attribute set to `false`. By default it is set to `true` and the element is not necessary if the user wishes to have this default behavior.

<transport name="name" pivot="handler type">
Defines a transport on the client side, which is invoked when sending a SOAP message. The "pivot" attribute specifies a Handler to be used as the actual sender for this transport (for example, the HTTPSender). Request and response flows may be specified as in server-side transports to do processing on the request (i.e. outgoing message) or response (i.e. incoming message).

<typeMapping qname="ns:localName" type="java:classname" serializer="classname" deserializer="classname"/>
Each typeMapping maps an XML qualified name to/from a Java class, using a specified Serializer and Deserializer.

<beanMapping qname="ns:localName" type="java:classname">
A simplified type mapping, which uses pre-defined serializers/deserializers to

encode/decode JavaBeans. The class named by "classname" must follow the JavaBean standard pattern of get/set accessors.

<arrayMapping qname="ns:localName" type="java:classname" innerType="ns:innerTypeQName">

A specialized type mapping, which uses pre-defined serializers/deserializers to encode/decode Arrays. The class named by "classname" must be an array type (ie name ending with "[]").

<documentation>

Can be used inside a <service>, an <operation> or an operation <parameter>. The content of the element is arbitrary text which will be put in the generated wsdl inside a wsdl:document element.

Example:

```
<operation name="echoString" >
  <documentation>This operation echoes a string</documentation>
  <parameter name="param">
    <documentation>a string</documentation>
  </parameter>
</operation>
```

1.4. Global Axis Configuration

The server is configured (by default) by values in the server-config.wsdd file, though a dedicated Axis user can write their own configuration handler, and so store configuration data in an LDAP server, database, remote web service, etc. Consult the source on details as to how to do that. You can also add options to the web.xml file and have them picked up automatically. We don't encourage that as it is nice to keep configuration stuff in one place.

In the server-config file, there is a global configuration section, which supports parameter name/value pairs as nested elements. Here are the options that we currently document, though there may be more (consult the source, as usual).

```
<globalConfiguration>
  <parameter name="adminPassword" value="admin"/>
  <parameter name="attachments.Directory" value="c:\temp\attachments"/>
  <parameter name="sendMultiRefs" value="true"/>
  <parameter name="sendXsiTypes" value="true"/>
  <parameter name="attachments.implementation"
             value="org.apache.axis.attachments.AttachmentsImpl"/>
  <parameter name="sendXMLDeclaration" value="true"/>
  <parameter name="enable2DArrayEncoding" value="true"/>
  <parameter name="dotNetSoapEncFix" value="false"/>
</globalConfiguration>
```

1.5. Individual Service Configuration

TODO

Here are some of the per-service configuration options are available; these can be set in the wsdd file used to deploy a service, from where they will be picked up.

More may exist.

style	whether to use RPC:enc or doc/lit encoding
SingleSOAPVersion	When set to either "1.1" or "1.2", this configures a service to only accept the specified SOAP version. Attempts to connect to the service using another version will result in a fault.
wsdlFile	The path to a WSDL File; can be an absolute path or a resource that axis.jar can load. Useful to export your custom WSDL file. When specify a path to a resource, place a forward slash to start at the beginning of the classpath (e.g "/org/someone/res/mywsdl.wsdl"). How does Axis know whether to return a file or resource? It looks for a file first, if that is missing a resource is returned.

1.6. Axis Logging Configuration

Axis uses the Jakarta Projects's commons-logging API, as implemented in `commons-logging.jar` to implement logging throughout the code. Normally this library routes the logging to the Log4j library, provided that an implementation of log4j is on the classpath of the server or client. The commons-logging API can also bind to Avalon, `System.out` or the Java1.4 logger. The JavaDocs for the library explain the process for selecting a logger, which can be done via a system property or a properties file in the classpath.

Log4J can be configured using the file log4j.properties in the classpath; later versions also support an XML configuration. Axis includes a preconfigured log4j.properties file in `axis.jar`. While this is adequate for basic use, any complex project will want to modify their own version of the file. Here is what to do

1. Open up axis.jar in a zipfile viewer and remove log4j.properties from the jar
2. Or, when building your own copy of axis.jar, set the Ant property `exclude.log4j.configuration` to keep the properties file out the JAR.
3. Create your own log4J.properties file, and include it in `WEB-INF/classes` (server-side), in your main application JAR file client side.
4. Edit this log4J properties file to your hearts content. Server side, setting up rolling logs with fancy html output is convenient, though once you start clustering the back end servers that ceases to be as usuable. Log4J power tools, such as 'chainsaw', are the secret

here.

1.6.1. Log Categories

Axis classes that log information create their own per-class log, each of which may output information at different levels. For example, the main entry point servlet has a log called `org.apache.axis.transport.http.AxisServlet`, the AxisEngine is `org.apache.axis.AxisEngine`, and so on. There are also special logs for special categories.

`org.apache.axis.TIME`	A log that records the time to execute incoming messages, splitting up into preamble, invoke, post and send times. These are only logged at debug level.
`org.apache.axis.EXCEPTIONS`	Exceptions that are sent back over the wire. AxisFaults, which are normally created in 'healthy' operation, are logged at debug level. Other Exceptions are logged at the Info level, as they are more indicative of server side trouble.
`org.apache.axis.enterprise`	"Enterprise" level stuff, which generally means stuff that an enterprise product might want to track, but in a simple environment (like the Axis build) would be nothing more than a nuisance.

1.7. Pre-Configured Axis Components Reference

1.7.1. On the server

SimpleSessionHandler
uses SOAP headers to do simple session management
LogHandler
The LogHandler will simply log a message to a logger when it gets invoked.
SoapMonitorHandler
Provides the hook into the message pipeline sending the SOAP request and response messages to the SoapMonitor utility.
DebugHandler
Example handler that demonstrates dynamically setting the debug level based on a the value of a soap header element.
ErrorHandler
Example handler that throws an AxisFault to stop request/response flow processing.
EchoHandler

The EchoHandler copies the request message into the response message.

HTTPAuth

The HTTPAuthHandler takes HTTP-specific authentication information (right now, just Basic authentication) and turns it into generic MessageContext properties for username and password

SimpleAuthenticationHandler

The SimpleAuthentication handler passes a MessageContext to a SecurityProvider (see org.apache.axis.security) to authenticate the user using whatever information the SecurityProvider wants (right now, just the username and password).

SimpleAuthorizationHandler

This handler, typically deployed alongside the SimpleAuthenticationHandler (a chain called "authChecks" is predefined for just this combination), checks to make sure that the currently authenticated user satisfies one of the allowed roles for the target service. Throws a Fault if access is denied.

MD5AttachHandler

Undocumented, uncalled, untested handler that generates an MD5 hash of attachment information and adds the value as an attribute in the soap body.

URLMapper

The URLMapper, an HTTP-specific handler, usually goes on HTTP transport chains (it is deployed by default). It serves to do service dispatch based on URL - for instance, this is the Handler which allows URLs like http://localhost:8080/axis/services/MyService?wsdl to work.

RPCProvider

The RPCProvider is the pivot point for all RPC services. It accepts the following options:
className = the class of the backend object to invoke
methodName = a space-separated list of methods which are exported as web services. The special value "*" matches all public methods in the class.

MsgProvider

The MsgProvider is the pivot point for all messaging services. It accepts the following options:
className = the class of the backend object to invoke
methodName = a space-separated list of methods which are exported as web services. The special value "*" matches all public methods in the class.

JWSHandler

Performs drop-in deployment magic.

JAXRPCHandler

Wrapper around JAX-RPC compliant handlers that exposes an Axis handler

interface to the engine.

LocalResponder

The LocalResponder is a Handler whose job in life is to serialize the response message coming back from a local invocation into a String. It is by default on the server's local transport response chain, and it ensures that serializing the message into String form happens in the context of the server's type mappings.

1.7.2. On the client

SimpleSessionHandler

uses SOAP headers to do simple session management

JAXRPCHandler

Wrapper around JAX-RPC compliant handlers that exposes an Axis handler interface to the engine.

HTTPSender

A Handler which sends the request message to a remote server via HTTP, and collects the response message.

LocalSender

A Handler which sends the request message to a "local" AxisServer, which will process it and return a response message. This is extremely useful for testing, and is by default mapped to the "local:" transport. So, for instance, you can test the AdminClient by doing something like this:

```
% java org.apache.axis.client.AdminClient -llocal:// list
```