# WebServices - Axis

**1. Axis installation instructions**

## 1.1. Table of Contents



## 1.2. Introduction

This document describes how to install Apache Axis. It assumes you already know how to write and run Java code and are not afraid of XML. You should also have an application server or servlet engine and be familiar with operating and deploying to it. If you need an

application server, we recommend Jakarta Tomcat. [If you are installing Tomcat, get the latest 4.1.x version, and the full distribution, not the LE version for Java 1.4, as that omits the Xerces XML parser]. Other servlet engines are supported, provided they implement version 2.2 or greater of the servlet API. Note also that Axis client and server requires Java 1.3 or later.

For more details on using Axis, please see the user guide.

## 1.3. Things you have to know

A lot of problems with Axis are encountered by people who are new to Java, server-side Java and SOAP. While you can learn about SOAP as you go along, writing Axis clients and servers is not the right time to be learning foundational Java concepts, such as what an array is, or basic application server concepts such as how servlets work, and the basics of the HTTP protocol.

Things you need to know before writing a Web Service:

1. Core Java datatypes, classes and programming concepts.
2. What threads are, race conditions, thread safety and sychronization.
3. What a classloader is, what hierarchical classloaders are, and the common causes of a "ClassNotFoundException".
4. How to diagnose trouble from exception traces, what a NullPointerException (NPE) and other common exceptions are, and how to fix them.
5. What a web application is; what a servlet is, where classes, libraries and data go in a web application.
6. How to start your application server and deploy a web application on it.
7. What a network is, the core concepts of the IP protocol suite and the sockets API. Specifically, what is TCP/IP.
8. What HTTP is. The core protocol and error codes, HTTP headers and perhaps the details of basic authentication.
9. What XML is. Not necessarily how to parse it or anything, just what constitutes well-formed and valid XML.

Axis and SOAP depends on all these details. If you don't know them, Axis (or anyone else's Web Service middleware) is a dangerous place to learn. Sooner or later you will be forced to discover these details, and there are easier places to learn than Axis.

If you are completely new to Java, we recommend you start off with things like the Java Tutorials on Sun's web site, and perhaps a classic book like Thinking in Java, until you have enough of a foundation to be able to work with Axis. It is also useful to have written a simple web application, as this will give you some knowledge of how HTTP works, and how Java application servers integrate with HTTP. You may find the course notes from Mastering the

World Wide Web useful in this regard, even though Axis is only introduced in lecture 28.

Be aware that there is a lot more needed to be learned in order to use Axis and SOAP effectively than the listing above. The other big area is "how to write internet scale distributed applications". Nobody knows how to do that properly yet, so that you have to learn this by doing.

## 1.4. Step 0: Concepts

Apache Axis is an Open Source SOAP server and client. SOAP is a mechanism for inter-application communication between systems written in arbitrary languages, across the Internet. SOAP usually exchanges messages over HTTP: the client POSTs a SOAP request, and receives either an HTTP success code and a SOAP response or an HTTP error code. Open Source means that you get the source, but that there is no formal support organisation to help you when things go wrong.

SOAP messages are XML messages. These messages exchange structured information between SOAP systems. Messages consist of one or more SOAP elements inside an envelope, Headers and the SOAP Body. SOAP has two syntaxes for describing the data in these elements, Section 5, which is a clear descendant of the XML RPC system, and XML Schema, which is the newer (and usually better) system. Axis handles the magic of converting Java objects to SOAP data when it sends it over the wire or receives results. SOAP Faults are sent by the server when something goes wrong; Axis converts these to Java exceptions.

SOAP is intended to link disparate systems. It is not a mechanism to tightly bind Java programs written by the same team together. It can bind Java programs together, but not as tightly as RMI or Corba. If you try sending many Java objects that RMI would happily serialize, you will be disappointed at how badly Axis fails. This is by design: if Axis copied RMI and serialized Java objects to byte streams, you would be stuck to a particular version of Java everywhere.

Axis implements the JAX-RPC API, one of the standard ways to program Java services. If you look at the specification and tutorials on Sun's web site, you will understand the API. If you code to the API, your programs will work with other implementations of the API, such as those by Sun and BEA. Axis also provides extension features that in many ways extends the JAX-RPC API. You can use these to write better programs, but these will only work with the Axis implementation. But since Axis is free and you get the source, that should not matter.

Axis is compiled in the JAR file axis.jar; it implements the JAX-RPC API declared in the JAR files jaxrpc.jar and saaj.jar. It needs various helper libraries, for logging, WSDL

processing and introspection. All these files can be packaged into a web application, axis.war, that can be dropped into a servlet container. Axis ships with some sample SOAP services. You can add your own by adding new compiled classes to the Axis webapp and registering them.

Before you can do that, you have to install it and get it working.

## 1.5. Step 1: Preparing the webapp

Here we assume that you have a web server up and running on the localhost at port 8080. If your server is on a different port, replace references to 8080 to your own port number.

In your Application Server installation, you should find a directory into which web applications ("webapps") are to be placed. Into this directory copy the webapps/axis directory from the xml-axis distribution. You can actually name this directory anything you want, just be aware that the name you choose will form the basis for the URL by which clients will access your service. The rest of this document assumes that the default webapp name, "axis" has been used; rename these references if appropriate.

## 1.6. Step 2: Setting up the libraries

In the Axis directory, you will find a WEB-INF sub-directory. This directory contains some basic configuration information, but can also be used to contain the dependencies and web services you wish to deploy.

Axis needs to be able to find an XML parser. If your application server or Java runtime does not make one visible to web applications, you need to download and add it. Java 1.4 includes the Crimson parser, so you can omit this stage, though the Axis team prefer Xerces.

To add an XML parser, acquire the JAXP 1.1 XML compliant parser of your choice. We recommend Xerces jars from the xml-xerces distribution, though others mostly work. Unless your JRE or app server has its own specific requirements, you can add the parser's libraries to axis/WEB-INF/lib. The examples in this guide use Xerces. This guide adds xml-apis.jar and xercesImpl.jar to the AXISCLASSPATH so that Axis can find the parser (see below).

If you get ClassNotFound errors relating to Xerces or DOM then you do not have an XML parser installed, or your CLASSPATH (or AXISCLASSPATH) variables are not correctly configured.

### 1.6.1. Tomcat 4.x and Java 1.4

Java 1.4 changed the rules as to how packages beginning in java.* and javax.* get loaded.

Specifically, they only get loaded from endorsed directories. jaxrpc.jar and saaj.jar contain javax packages, so they may not get picked up. If happyaxis.jsp (see below) cannot find the relevant packages, copy them from axis/WEB-INF/lib to CATALINA_HOME/common/lib and restart Tomcat.

### 1.6.2. WebLogic 8.1

WebLogic 8.1 ships with `webservices.jar` that conflicts with Axis' `saaj.jar` and prevents Axis 1.2 from working right out of the box. This conflict exists because WebLogic uses an older definition of `javax.xml.soap.*` package from Java Web Services Developer Pack Version 1.0, whereas Axis uses a newer revision from J2EE 1.4.

However, there are two alternative configuration changes that enable Axis based web services to run on Weblogic 8.1.

- In a webapp containing Axis, set <prefer-web-inf-classes> element in `WEB-INF/weblogic.xml` to true. An example of `weblogic.xml` is shown below:

```
<weblogic-web-app>
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>
```

  If set to `true`, the `<prefer-web-inf-classes>` element will force WebLogic's classloader to load classes located in the WEB-INF directory of a web application in preference to application or system classes. This is a recommended approach since it only impacts a single web module.

- In a script used to start WebLogic server, modify `CLASSPATH` property by placing Axis's `saaj.jar` library in front of WebLogic's `webservices.jar`.

  NOTE: This approach impacts all applications deployed on a particular WebLogic instance and may prevent them from using WebLogic's webservices.

For more information on how WebLogic's class loader works, see WebLogic Server Application Classloading.

## 1.7. Step 3: starting the web server

This varies on a product-by-product basis. In many cases it is as simple as double clicking on a startup icon or running a command from the command line.

## 1.8. Step 4: Validate the Installation

After installing the web application and dependencies, you should make sure that the server

is running the web application.

### 1.8.1. Look for the start page

Navigate to the start page of the webapp, usually http://127.0.0.1:8080/axis/, though of course the port may differ.

You should now see an Apache-Axis start page. If you do not, then the webapp is not actually installed, or the appserver is not running.

### 1.8.2. Validate Axis with happyaxis

Follow the link Validate the local installation's configuration
This will bring you to happyaxis.jsp a test page that verifies that needed and optional libraries are present. The URL for this will be something like http://localhost:8080/axis/happyaxis.jsp

If any of the needed libraries are missing, Axis will not work.
You must not proceed until all needed libraries can be found, and this validation page is happy.
Optional components are optional; install them as your need arises. If you see nothing but an internal server error and an exception trace, then you probably have multiple XML parsers on the CLASSPATH (or AXISCLASSPATH), and this is causing version confusion. Eliminate the extra parsers, restart the app server and try again.

### 1.8.3. Look for some services

From the start page, select View the list of deployed Web services. This will list all registered Web Services, unless the servlet is configured not to do so. On this page, you should be able to click on (wsdl) for each deployed Web service to make sure that your web service is up and running.

Note that the 'instant' JWS Web Services that Axis supports are not listed in this listing here. The install guide covers this topic in detail.

### 1.8.4. Test a SOAP Endpoint

Now it's time to test a service. Although SOAP 1.1 uses HTTP POST to submit an XML request to the endpoint, Axis also supports a crude HTTP GET access mechanism, which is useful for testing. First let's retrieve the version of Axis from the version endpoint, calling the getVersion method:

http://localhost:8080/axis/services/Version?method=getVersion

This should return something like:

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getVersionResponse
     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <getVersionReturn
       xsi:type="xsd:string">
         Apache Axis version: 1.1 Built on Apr 04, 2003 (01:30:37 PST)
      </getVersionReturn>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The Axis version and build date may of course be different.

### 1.8.5. Test a JWS Endpoint

Now let's test a JWS web service. Axis' JWS Web Services are java files you save into the Axis webapp anywhere but the WEB-INF tree, giving them the .jws extension. When someone requests the .jws file by giving its URL, it is compiled and executed. The user guide covers JWS pages in detail.

To test the JWS service, we make a request against a built-in example, EchoHeaders.jws (look for this in the axis/ directory).

Point your browser at http://localhost:8080/axis/EchoHeaders.jws?method=list.

This should return an XML listing of your application headers, such as

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <listResponse
     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <listReturn xsi:type="soapenc:Array"
       soapenc:arrayType="xsd:string[6]"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        <item>accept:image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*</item>
        <item>accept-language:en-us</item>
        <item>accept-encoding:gzip, deflate</item>
        <item>user-agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)</item>
        <item>host:localhost:8080</item>
        <item>connection:Keep-Alive</item>
      </listReturn>
```

```
    </listResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Again, the exact return values will be different, and you may need to change URLs to correct any host, port and webapp specifics.

## 1.9. Step 5: Installing new Web Services

So far you have got Axis installed and working--now it is time to add your own Web Service.

The process here boils down to (1) get the classes and libraries of your new service into the Axis WAR directory tree, and (2) tell the AxisEngine about the new file. The latter is done by submitting an XML deployment descriptor to the service via the Admin web service, which is usually done with the AdminClient program or the <axis-admin> Ant task. Both of these do the same thing: they run the Axis SOAP client to talk to the Axis administration service, which is a SOAP service in its own right. It's also a special SOAP service in one regard--it is restricted to local callers only (not remote access) and is password protected to stop random people from administrating your service. There is a default password that the client knows; if you change it then you need to pass the new password to the client.

The first step is to add your code to the server.

In the WEB-INF directory, look for (or create) a "classes" directory (i.e. axis/WEB-INF/classes ). In this directory, copy the compiled Java classes you wish to install, being careful to preserve the directory structure of the Java packages.

If your classes services are already packaged into JAR files, feel free to drop them into the WEB-INF/lib directory instead. Also add any third party libraries you depend on into the same directory.

After adding new classes or libraries to the Axis webapp, you must restart the webapp. This can be done by restarting your application server, or by using a server-specific mechanism to restart a specific webapp.

Note: If your web service uses the simple authorization handlers provided with xml-axis (this is actually not recommended as these are merely illustrations of how to write a handler than intended for production use), then you will need to copy the corresponding users.lst file into the WEB-INF directory.

## 1.10. Step 6: Deploying your Web Service

The various classes and JARs you have just set up implement your new Web Service. What remains to be done is to tell Axis how to expose this web service. Axis takes a Web Service

Deployment Descriptor (WSDD) file that describes in XML what the service is, what methods it exports and other aspects of the SOAP endpoint.

The users guide and reference guide cover these WSDD files; here we are going to use one from the Axis samples: the stock quote service.

### 1.10.1. Classpath setup

In order for these examples to work, java must be able to find axis.jar, commons-discovery.jar, commons-logging.jar, jaxrpc.jar, saaj.jar, log4j-1.2.8.jar (or whatever is appropriate for your chosen logging implementation), and the XML parser jar file or files (e.g., xerces.jar). These examples do this by adding these files to AXISCLASSPATH and then specifying the AXISCLASSPATH when you run them. Also for these examples, we have copied the xml-apis.jar and xercesImpl.jar files into the AXIS_LIB directory. An alternative would be to add your XML parser's jar file directly to the AXISCLASSPATH variable or to add all these files to your CLASSPATH variable.

On Windows, this can be done via the following. For this document we assume that you have installed Axis in C:\axis. To store this information permanently in WinNT/2000/XP you will need to right click on "My Computer" and select "Properties". Click the "Advanced" tab and create the new environmental variables. It is often better to use WordPad to create the variable string and then paste it into the appropriate text field.

```
set AXIS_HOME=c:\axis
set AXIS_LIB=%AXIS_HOME%\lib
set AXISCLASSPATH=%AXIS_LIB%\axis.jar;%AXIS_LIB%\commons-discovery.jar;
  %AXIS_LIB%\commons-logging.jar;%AXIS_LIB%\jaxrpc.jar;%AXIS_LIB%\saaj.jar;
  %AXIS_LIB%\log4j-1.2.8.jar;%AXIS_LIB%\xml-apis.jar;%AXIS_LIB%\xercesImpl.jar
```

Unix users have to do something similar. Below we have installed AXIS into /usr/axis and are using the bash shell. See your shell's documentation for differences. To make variables permeate you will need to add them to your shell's startup (dot) files. Again, see your shell's documentation.

```
set AXIS_HOME=/usr/axis
set AXIS_LIB=$AXIS_HOME/lib
set AXISCLASSPATH=$AXIS_LIB/axis.jar:$AXIS_LIB/commons-discovery.jar:
  $AXIS_LIB/commons-logging.jar:$AXIS_LIB/jaxrpc.jar:$AXIS_LIB/saaj.jar:
  $AXIS_LIB/log4j-1.2.8.jar:$AXIS_LIB/xml-apis.jar:$AXIS_LIB/xercesImpl.jar
export AXIS_HOME; export AXIS_LIB; export AXISCLASSPATH
```

To use Axis client code, you can select AXISCLASSPATH when invoking Java by entering

```
java -cp %AXISCLASSPATH% ...
```

or

```
java -cp "$AXISCLASSPATH" ...
```

depending on the platform. You may omit the quotes if your CLASSPATH doesn't have

spaces in it.

Also, it is probably a good time to add the AXISCLASSPATH variable to your CLASSPATH variable. This will enable you to not include the AXISCLASSPATH variable when launching the examples in this guide. This document assumes that you have NOT done this.

### 1.10.2. Find the deployment descriptor

Look in axis/samples/stock for the file deploy.wsdd. This is the deployment descriptor we want to tell Axis about. Deployment descriptors are an Axis-specific XML file that tells Axis how to deploy (or undeploy) a Web Service, and how to configure Axis itself. The Axis Administration Web Service lets the AdminClient program and its Ant task counterpart submit a new WSDD file for interpretation. The Axis 'engine' will update its configuration, then save its state.

By default Axis saves it state into the global configuration file axis/WEB-INF/server-config.wsdd. Sometimes you see a warning message about such a file not being found--don't worry about this, because Axis auto-creates the file after you deploy something to it. You can check in the webapp to see what this file looks like--and even copy it to other systems if you want to give them identical configurations. Note that Axis needs an expanded web application and write access to the WEB-INF dir to save its state in this location.

### 1.10.3. Run the admin client

Execute the following command from the samples/stock directory. If you are not in this directory you will get a "java.io.FileNotFoundException: deploy.wsdd (The system cannot find the file specified)" exception.

On Windows
```
java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient
     -lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```
On UNIX
```
java -cp $AXISCLASSPATH org.apache.axis.client.AdminClient
     -lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

If you get some java client error (like ClassNotFoundException), then you haven't set up your AXISCLASSPATH (or CLASSPATH) variable right, mistyped the classname, or did some other standard error. Tracking down such problems are foundational Java development skills--if you don't know how to do these things, learn them now!

Note: You may need to replace localhost with your host name, and 8080 with the port number used by your web server. If you have renamed the web application to something

other than "axis" change the URL appropriately.

If you get some AxisFault listing, then the client is working, but the deployment was unsuccessful. This is where the knowledge of the sockets API to TCP and the basics of the HTTP that Web Service development requires begins to be needed. If you got some socket error like connection refused, the computer at the far end isn't talking to you, so find the cause of that and fix it. If you get an HTTP error code back find out what the error means and correct the problem. These skills are fundamental to using web services.

The user's guide covers the AdminClient in more detail, and there is also an Ant task to automate the use of Axis in your Ant build scripts.

## 1.11. Step 7: Testing

This step is optional, but highly recommended. For illustrative purposes, it is presumed that you have installed and deployed the stock quote demo.

- Change directory to the distribution directory for xml-axis and execute the following command (or its Unix equivalent):

On Windows
```
java -cp .;%AXISCLASSPATH% samples.stock.GetQuote
     -lhttp://localhost:8080/axis/servlet/AxisServlet
     -uuser1 -wpass1 XXX
```
On UNIX
```
java -cp $AXISCLASSPATH samples.stock.GetQuote
     -lhttp://localhost:8080/axis/servlet/AxisServlet
     -uuser1 -wpass1 XXX
```

- You should get back "55.25" as a result.

Note: Again, you may need to replace localhost with your host name, and 8080 with the port number used by your web server. If you have renamed the web application to something other than "axis" change the URL appropriately.

## 1.12. Advanced Installation: adding Axis to your own Webapp

If you are experienced in web application development, and especially if you wish to add web services to an existing or complex webapp, you can take an alternate approach to running Axis. Instead of adding your classes to the Axis webapp, you can add Axis to your application.

The core concepts are

1. Add axis.jar, wsdl.jar, saaj.jar, jaxrpc.jar and the other dependent libraries to your WAR file.

2. Copy all the Axis Servlet declarations and mappings from axis/WEB-INF/web.xml and add them to your own web.xml
3. Build and deploy your webapp.
4. Run the Axis AdminClient against your own webapp, instead of Axis, by changing the URL you invoke it with.

The process is also covered in Chapter 15 of Java Development with Ant, which can be downloaded as a PDF file.

## 1.13. What if it doesn't work?

Axis is a complicated system to install. This is because it depends on the underlying functionality of your app server, has a fairly complex configuration, and, like all distributed applications, depends upon the network too.

We see a lot of people posting their problems on the axis-user mailing list, and other Axis users as well as the Axis developers do their best to help when they can. But before you rush to post your own problems to the mailing list, a word of caution:

Axis is free. This means nobody gets paid to man the support lines. All the help you get from the community is voluntary and comes from the kindness of their hearts. They may be other users, willing to help you get past the same hurdles they had to be helped over, or they may be the developers themselves. But it is all voluntary, so you may need to keep your expectations low!

1. Post to the user mail list, not the developer list. You may think the developer mail list is a short cut to higher quality answers. But the developers are also on the user list along with many other skilled users--so more people will be able to answer your questions. Also, it is helpful for all user issues to be on one list to help build the searchable mailing list archive.
2. Don't ask non-Axis-related questions. The list is not the place to ask about non-Axis, non-SOAP, problems. Even questions about the MS Soap toolkit or .NET client side, don't get many positive answers--we avoid them. That also goes for the Sun Java Web Services Developer Pack, or the Jboss.net stuff that they've done with Axis.
3. Never bother posting to the soapbuilders mailing list either, that is only for people developing SOAP toolkits, not using them--all off-topic messages are pointedly ignored.
4. There is no guarantee that anyone will be able to solve your problem. The usual response in such a situation is silence, for a good reason: if everybody who didn't know the answer to a question said "I don't know", the list would be overflowed with noise. Don't take silence personally.
5. Never expect an immediate answer. Even if someone knows the answer, it can take a day or two before they read their mail. So if you don't get an answer in an hour or two, don't panic and resend. Be patient. And put the time to use by trying to solve your problems

yourself.

6. Do your homework first. This document lists the foundational stuff you need to understand. It has also warned you that it can take a day to get a reply. Now imagine you get a HTTP Error '404' on a SOAP call. Should you rush to post a 'help' request, or should you try and find out what an HTTP error code is, what #404 usually means and how to use a Java debugger. We provide the source to make that debugging easier :)
7. Post meaningful subject lines. You want your message read, not deleted unread. A subject line of 'Axis problem', 'Help with Axis', etc. is not meaningful, and is not likely to get many readers.
8. Search the mailing list archives FIRST to see if someone had the same problem. This list is searchable--and may save you much time in getting an answer to your problem.
9. Use the jira database to search for Axis bugs, both open and closed.
10. Consult the Axis Wiki for its Frequently Asked Questions (FAQ), installation notes, interoperability issues lists, and other useful information.
11. Don't email people for help directly, unless you know them. It's rude and presumptuous. Messages sent over the mail list benefit the whole community--both the original posters and people who search the list. Personal messages just take up the recipients time, and are unwelcome. Usually, if not ignored outright, recipients of personal requests will just respond 'ask the mail list' anyway!
12. Know that configuration problems are hard to replicate, and so can be difficult to get help on. We have tried with the happyaxis.jsp demo to automate the diagnostics gathering for you, but it can be hard for people to be of help here, especially for obscure platforms.
13. Keep up to date with Axis releases, even the beta copies of forthcoming releases. You wouldn't want your problem to be a bug that was already known and fixed in a more recent release. Often the common response to any question is 'have you tried the latest release'.
14. Study and use the source, and fix it when you find defects. Even fix the documentation when you find defects. It is only through the participation of Axis' users that it will ever get better.

Has this put you off joining and participating in the Axis user mail list? We hope not--this list belongs to the people who use Axis and so will be your peers as your project proceeds. We just need for you to be aware that it is not a 24x7 support line for people new to server side Java development, and that you will need to be somewhat self sufficient in this regard. It is not a silver bullet. However, knowing how to make effective use of the list will help you develop better with Axis.

## 1.14. Summary

Axis is simply an implementation of SOAP which can be added to your own webapp, and a webapp which can host your own web services. Installing it can be a bit fiddly, especially

given Java 1.4's stricter requirements. If you follow a methodical process, including testing along the way, using happyaxis and the bundled test services, you will find it easier to get started with Axis.

## 1.15. Appendix: Enabling the SOAP Monitor

SOAP Monitor allows for the monitoring of SOAP requests and responses via a web browser with Java plug-in 1.3 or higher. For a more comprehensive explanation of its usage, read Using the SOAP Monitor in the User's Guide.

By default, the SOAP Monitor is not enabled. The basic steps for enabling it are compiling the SOAP Monitor java applet, deploying the SOAP Monitor web service and adding request and response flow definitions for each monitored web service. In more detail:

1. Go to $AXIS_HOME/webapps/axis (or %AXIS_HOME%\webapps\axis) and compile SOAPMonitorApplet.java.

   On Windows
   ```
   javac -classpath %AXIS_HOME%\lib\axis.jar SOAPMonitorApplet.java
   ```
   On Unix
   ```
   javac -classpath $AXIS_HOME/lib/axis.jar SOAPMonitorApplet.java
   ```

   Copy all resulting class files (i.e. SOAPMonitorApplet*.class) to the root directory of the web application using the SOAP Monitor (e.g. .../tomcat/webapps/axis)

2. Deploy the SOAPMonitorService web service with the admin client and the deploy-monitor.wsdd file (shown below).

   Go to the directory deploy-monitor.wsdd is located and execute the command below. The command assume that /axis is the intended web application and it is available on port 8080.

   On Windows
   ```
   java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient
        -lhttp://localhost:8080/axis/services/AdminService deploy-monitor.wsdd
   ```
   On UNIX
   ```
   java -cp $AXISCLASSPATH org.apache.axis.client.AdminClient
        -lhttp://localhost:8080/axis/services/AdminService deploy-monitor.wsdd
   ```

   SOAPMonitorService Deployment Descriptor (deploy-monitor.wsdd)
   ```
   <deployment xmlns="http://xml.apache.org/axis/wsdd/"
               xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
     <handler name="soapmonitor"
              type="java:org.apache.axis.handlers.SOAPMonitorHandler">
       <parameter name="wsdlURL"
                  value="/axis/SOAPMonitorService-impl.wsdl"/>
       <parameter name="namespace"
                  value="http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl"/
       <parameter name="serviceName" value="SOAPMonitorService"/>
   ```

```
    <parameter name="portName" value="Demo"/>
  </handler>
  <service name="SOAPMonitorService" provider="java:RPC">
    <parameter name="allowedMethods" value="publishMessage"/>
    <parameter name="className"
               value="org.apache.axis.monitor.SOAPMonitorService"/>
    <parameter name="scope" value="Application"/>
  </service>
</deployment>
```

3. For each service that is to be monitored, add request and response flow definitions to the service's deployment descriptor and deploy (or redeploy) the service. The requestFlow and responseFlow definitions follow the start tag of the <service> element. If a service is already deployed, undeploy it and deploy it with the modified deployment descriptor. An example is shown below:

```
...
   <service name="xmltoday-delayed-quotes" provider="java:RPC">
     <requestFlow>
       <handler type="soapmonitor"/>
     </requestFlow>
     <responseFlow>
       <handler type="soapmonitor"/>
     </responseFlow>
       ...
```

4. With a web browser, go to http[s]://host[:port][/webapp]/SOAPMonitor (e.g. http://localhost:8080/axis/SOAPMonitor) substituting the correct values for your web application. This will show the SOAP Monitor applet for viewing service requests and responses. Any requests to services that have been configured and deployed correctly should show up in the applet.