



The **Apache Software Foundation**
<http://www.apache.org/>

Web Services and SOA with AXIS 2

<http://axis.apache.org/axis2/java/core/index.html>

- ❖ Apache **Axis2** is the next generation **web service framework from Apache** software foundation.
- ❖ The Apache software foundation started Apache SOAP as its first web service framework.
- ❖ They developed Apache Axis, which became one of the very successful projects at Apache and is still used heavily in the industry.
- ❖ Apache **Web Service community** initiated **Apache Axis2 project in 2004**.
- ❖ Apache Axis2 has become the **de facto open source Java Web Service framework** and is now heavily
- ❖ used in both the **industry** and in **academia**.

Introduction (Wikipedia) <https://repository.apache.org/snapshots/org/apache/axis2/>

Apache Axis2 is a **core engine** for **Web services**. It is a complete re-design and re-write of the widely used **Apache Axis SOAP** stack. **Implementations of Axis2** are available **in Java** and **C**.

Axis2 not only provides the capability to add Web services interfaces to Web applications, but can **also function as a standalone server application**.

A new architecture for Axis2 was introduced^[by whom?] during the August 2004 Axis2 Summit in Colombo, Sri Lanka. The new architecture on which Axis2 is based is more flexible, efficient and configurable in comparison to Axis1.x architecture. Some well-established concepts from Axis 1.x, like handlers etc., have been preserved in the new architecture.

Apache Axis2 not only **supports** SOAP 1.1 and **SOAP 1.2**, but it also has integrated support for the widely popular **REST** style of Web services. The same business-logic implementation can offer both a **WS-*** style interface as well **as a REST/POX** style interface simultaneously.

Axis2/Java has support for **Spring Framework**.

Axis2/C seems to be **abandoned in 2009**.

Apache Axis2 includes support for **following standards**:

- **WS - ReliableMessaging** - Via **Apache Sandesha2**
- **WS - Coordination** - Via **Apache Kandula2**
- **WS - AtomicTransaction** - Via **Apache Kandula2**
- **WS - SecurityPolicy** - Via **Apache Rampart**
- **WS - Security** - Via **Apache Rampart**
- **WS - Trust** - Via **Apache Rampart**
- **WS - SecureConversation** - Via **Apache Rampart**
- **SAML 1.1** - Via **Apache Rampart**
- **SAML 2.0** - Via **Apache Rampart**

- [WS - Addressing](#) - Module included as part of Axis2 core Axis2 modules provides [QoS](#) features like security, reliable messaging, etc.
- [Apache Rampart module](#) - Apache Rampart modules adds [WS - Security](#) features
- Apache Sandesha module - An implementation of [WS - Reliable Messaging](#) specification

Web services and related technologies. **Topics:**

1. Service Oriented Architecture (SOA)
2. Web services overview
3. Web services standards and standard bodies
4. Apache Web Service stack
5. Getting started with Axis2

Ad. 1 Service Oriented Architecture (SOA)

- ❖ [RPC](#), [RMI](#), [IIOP](#), and [CORBA](#) are a few proposals that [provide abstractions](#) over the network [for the developers](#) to build upon. The goal of distributed computing is to provide such abstractions.
- ❖ [Service Oriented Architecture](#) (SOA) defines a set of [concepts](#) and [patterns](#) to [integrate](#) [homogenous](#) and [heterogeneous](#) components together.
- ❖ SOA provides a better way to achieve [loosely coupled systems](#), and hence more extensibility and flexibility.
- ❖ In addition, [similar to object-oriented programming](#) (OOP), SOA enables a high degree of [reusability](#).

There are three main ways one can [enable SOA capabilities](#) in their systems and applications

- [Existing messaging systems](#): for example, IBM MQSeries, Tibco, JMS
- [Plain Old XML](#) (POX): for example, XML/HTTP, REST, and so on
- [Web services](#): for example, [SOAP](#), [WSDL](#), [WS-*](#)

Ad. 2,3 Web services overview & standards bodies

- ❖ SOA is the concept where an [application is no longer a large monolithic program](#).
- ❖ It is [divided into smaller, loosely coupled programs](#).
- ❖ The provided [services are loosely coupled](#) together with standardized and [well-defined interfaces](#)

There are **three main standard bodies** that helped to improve the interoperability, quality of service, and base standards:

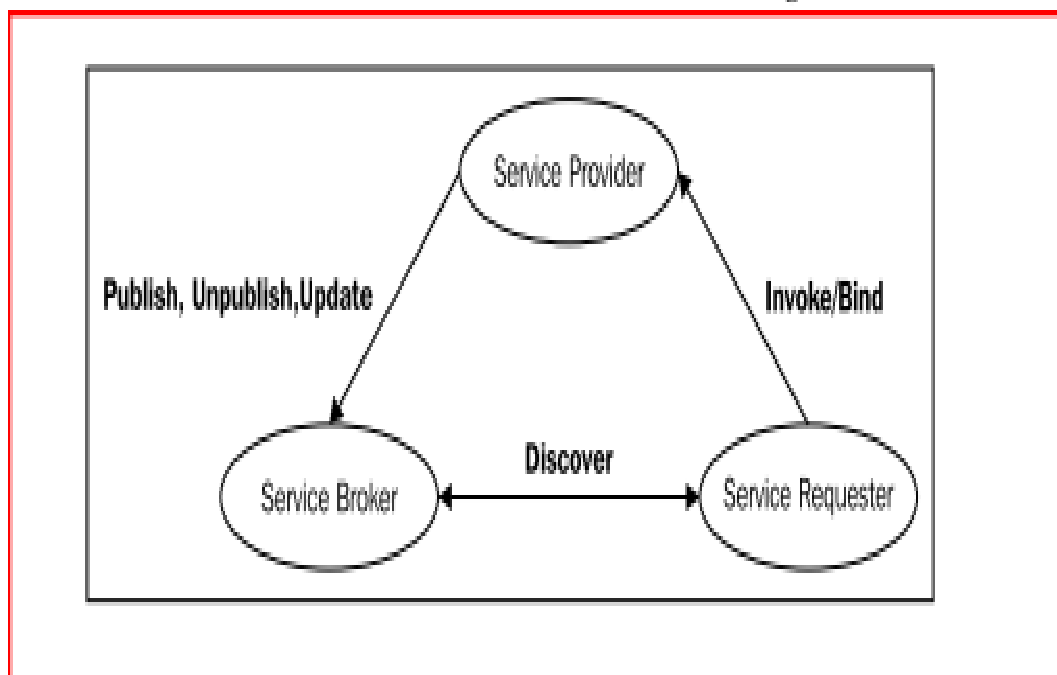
- **WS-I**
- **OASIS**
- **W3C**

The goal of **WS-I** is to create standards and procedures to enforce the required level of **interoperability** among various web service frameworks.

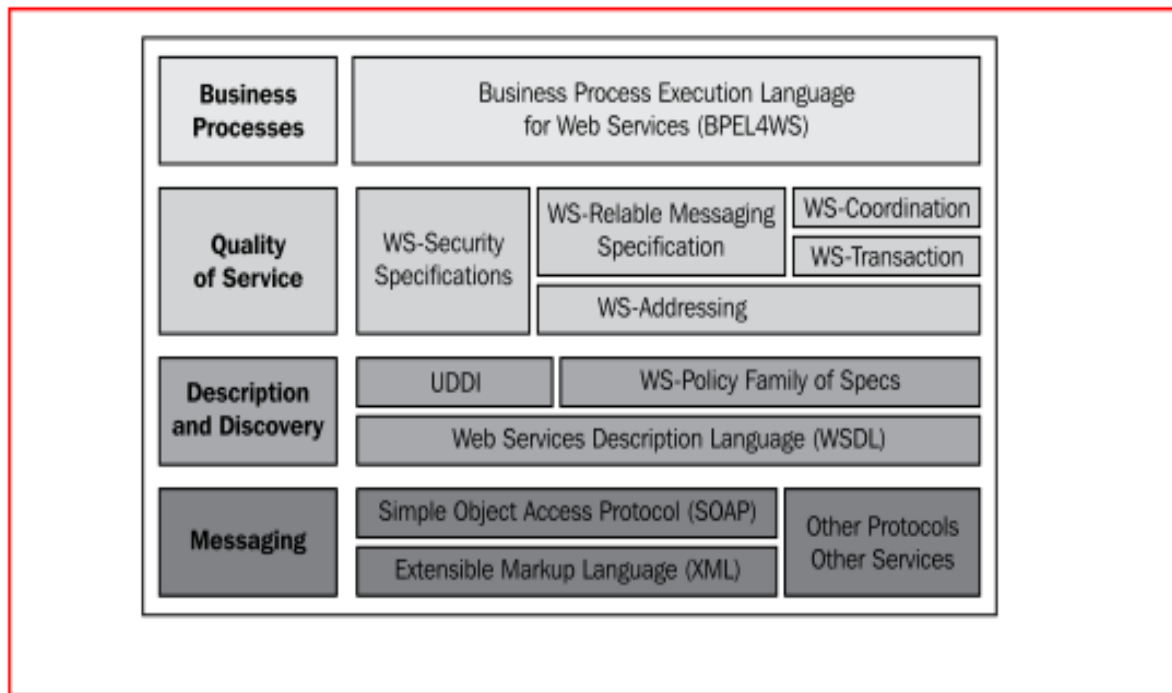
OASIS's main goal is to improve the **quality of services** of web services, which include security, reliability, transaction, and resource management.

W3C defines a **web service as a software system** designed to support interoperable machine-to-machine interaction over a network.

Web Services Model:

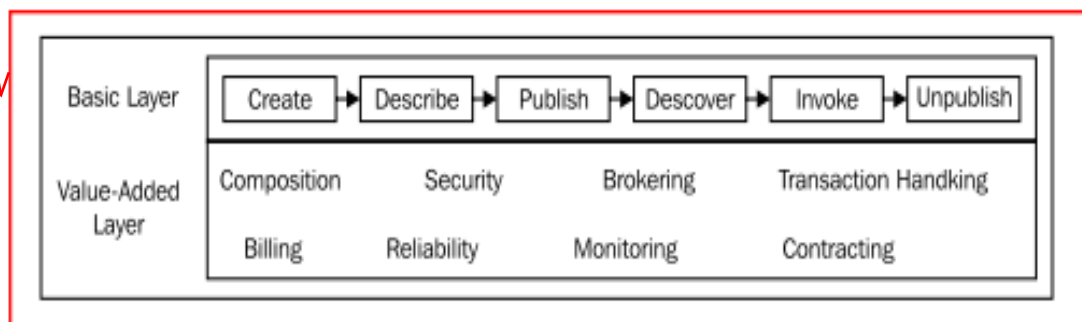


Web Services Standards:



Web services lifecycle:

- ❖ **Create:** The first activity in the service life cycle is the creation of the web service. This can be achieved either by building from scratch or by integrating existing web services.
- ❖ **Describe:** After creating the web service, it has to be described so that others can access it.
- ❖ **Publish:** After the description, it has to be published on the Web.
- ❖ **Discover:** Discovering a web service can be facilitated by a service broker, which will support requirement analysis and description of requester's need, matching needs to available web services, negotiation, and binding.
- ❖ As an alternative to discovery, often commercial agreements are made based on a supplied **WSDL**. This forms part of the contract between organizations.
- ❖ **Invoke:** Once the service is discovered, use tools and procedures to invoke the service.
- ❖ **Unpublish:** Finally, the service can be unpublished if it is no longer available or needed.



Ad 4. Apache Web Service stack

Apache SOAP - initially at IBM

Apache **Axis** (=Axis1) - based on SOAP with DOM as internal representation of XML

Apache **Axis2** – new XML processing framework = **Axiom**; support **SOAP 1.1, 1.2 & WS-*/REST**

Apache Axis2 is more efficient, more modular, and more XML-oriented than the older versions. Support the easy addition of **plugin "modules"**:

- **WS-ReliableMessaging**: supported by **Apache Sandesha2**
- **WS-Coordination** and **WS-Atomic Transaction**: supported by **Apache Kandula2**
- **WS-Security**: Supported by **Apache Rampart**
- **WS-Addressing**: module included as part of **Axis2 core**

Axis2 comes with **many new features**, enhancements, and industry specification implementations:

- Speed: Axis2 uses its own **object model** and **StAX** (Streaming API for XML) parsing to achieve significantly greater speed than earlier versions of Apache Axis.
- AXIOM: Axis2 comes with its own **lightweight object model**, **AXIOM**, for message processing, which is extensible, performs highly, and is developer-convenient.
- Hot deployment:
new services can be added to the system without **having to shut down the server**. Simply drop the required web service archive into the services directory in the repository, and the deployment model will automatically deploy the service and make it available for use.
- Asynchronous web services: Axis2 now supports **asynchronous web** services and asynchronous web services invocation using **non-blocking clients** and transports.
- MEP support: Axis2 now comes handy with the flexibility to support Message Exchange Patterns (MEPs) with in-built support for basic **MEPs defined in WSDL 2.0**.
- Flexibility: The Axis2 architecture gives the developer complete freedom to insert extensions into the engine for custom header processing, system management, and anything else that you can imagine.
- Stability: Axis2 defines a set of published interfaces, which change relatively slowly, as compared to the rest of Axis.
- Component-oriented deployment: You can easily define reusable networks of Handlers to implement common patterns of processing for your applications or to distribute to partners.

- Transport framework: Axis2 has a clean and simple abstraction for integrating and using Transports (that is, senders and listeners for SOAP over various protocols such as SMTP, FTP, message-oriented middleware, and so on), and the core of the engine is **completely transport-independent**.
- WSDL support: Axis2 supports the Web Service Description Language (versions 1.1 and 2.0), which **allows you to easily build stubs to access remote services**, and also to automatically export machine-readable descriptions of your deployed services from Axis2.
- Add-ons: Several web service specifications have been incorporated, including WSS4J for security (Apache Rampart), Sandesha for reliable messaging. Kandula is an encapsulation of WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity.
- Composition and extensibility: Modules and phases improve support for composability and extensibility. Modules support composability and can also **support new WS-* specifications** in a simple and clean manner. They are, however, not hot deployable, as they change the overall behavior of the system.

Installing AXIS2

Axis2 supports JDK 1.5 and higher.

There are three types of software:

- ❖ free software,
- ❖ open source software, and
- ❖ commercial software.

The download link:

http://ws.apache.org/axis2/download/1_5_1/download.cgi

Four different distributions:

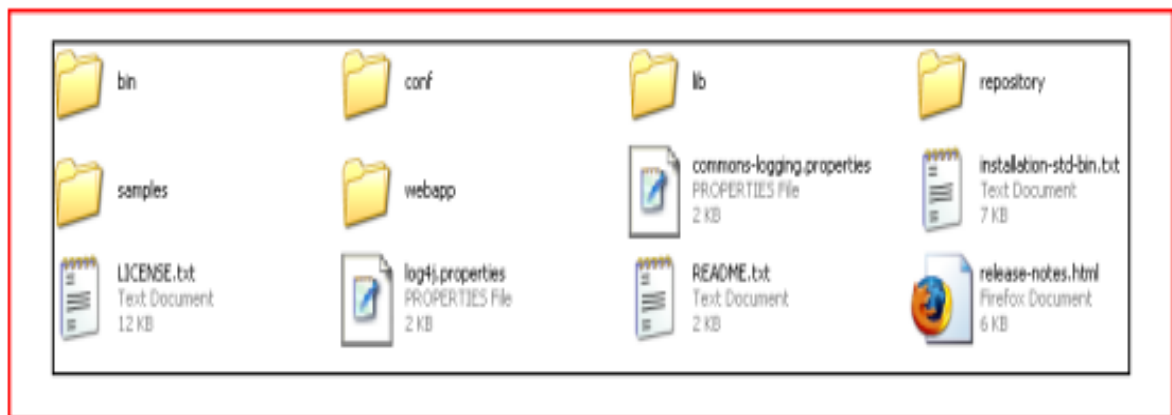
- Binary distribution
- WAR distribution
- Source distribution
- Document distribution

Ad. Binary distribution

- Axis2 **binary distribution** is a **complete package** where you can deploy services and expose them **using SimpleAxisServer**.
- **SimpleAxisServer** is a **fully functional server** that can be used as the backend server to expose the web service.
- It supports all the features that the **servlet** version supports, including **session** management, **thread** management, auto **WSDL generation**, and others.
- We can also use **Axis2 binary distribution** to **invoke remote services**.
- For this you need to add Axis2 and other related libraries into the class path and use Axis2 Client APIs to invoke the service.
- Binary distribution can also be used **to generate Stubs and Skeleton from a given WSDL** or to **generate WSDL from a given Java class**.
- Starting Axis2 as a standalone server is just a matter of running either .bat or a script file in the bin directory.
- Once we run **the axis2server.sh** (or .bat) and type **http://localhost:8080/axis2**, we can see the list of available **services** in the system, and these indicate the server is up and running.

Extract the binary distribution to a set of subdirectories:

bin, lib, samples, repository, webapp, and conf:



Ad. WAR distribution

- Assume you have already downloaded Axis2 **WAR distribution** & you **have Apache Tomcat** running.
- **To deploy Axis2**, you need to **copy the axis2.war** file into the **webapps** directory.
- **If Tomcat is running** on port 8080, you can access Axis2 by going to the following URL:
- **http://localhost:8080/axis2**



We can try to invoke the **version service** (a **default service** comes with Axis2 distribution) using the following URL:

<http://localhost:8080/axis2/services/Version/getVersion>

The answer:

```
<ns:getVersionResponse>
  <ns:return>Hi - the Axis2 version is SNAPSHOT</ns:return>
</ns:getVersionResponse>
```

Installing WAR distribution consists of the following few steps:

1. **Install application server:**

Apache Tomcat can be considered as one of the best application servers (does not support all)

2. Depending on the application server, you can find the location where you need to deploy the **WAR files**. If you take Tomcat as an example, you need to put the WAR file into the webapps directory. So let's drop Axis2 WAR distribution into the webapps directory of the application server.

3. As the final step, open a **browser** and type **<http://localhost:8080/axis2>**.

We can see Axis2 web application (as shown in the previous figure) home page (here the URL may differ, depending on the application server).

IDE. Axis2 JAR distribution provides a convenient way to download and embed Axis2 runtime into the IDE. You can download Axis2 library files separately or you can get those from Axis2 binary or WAR distribution:

<http://people.apache.org/repo/m2-ibiblio-archived/org/apache/axis>

Creating a Web Service

- ❖ **Code first approach:** It is the same as the **bottom-up approach**, where we first start with the source code and eventually expose our source code as a web service.
- ❖ The code first approach helps to easily convert an existing application into a web service.
- ❖ POJO or Plain Old Java Object is one of the very good examples of the code first approach.
- ❖ With the code first approach, you might not be able to get the full power of the web service.
- ❖ **Contract first approach:** the contract first approach is analogous to the **top-down approach**.
- ❖ In this process, we first write the Web Service Description Language (WSDL) document according to the service contract, its meaning, and so on.
- ❖ Once we have the WSDL document, we can use the tool supported by the web service framework to convert the WSDL document in the framework-dependent code (skeleton) and then complete the business logic.

The code first approach: Single class POJO approach

We will begin by writing a simple web service that says Hello <name> when invoked.

Using this approach, even someone who has no knowledge of WSDL or SOAP can deploy a service and easily access it as a web service. (HelloWorld.java)

```
public class HelloWorld {  
    public String sayHello(String name) {  
        return "Hello " + name;  
    }  
}
```

There are a number of ways of deploying a service.

Even for POJO deployment there are a few ways in which you can make Java class into a web service.

We assume that you are going to deploy the service in the Tomcat application server.

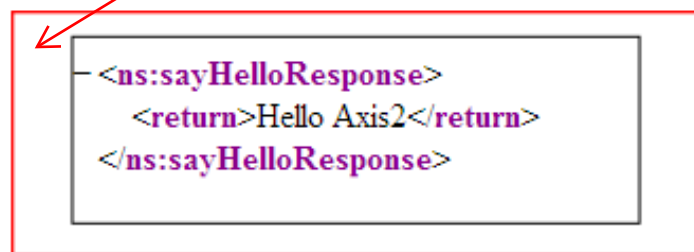
1. **Compile** the Java class, which will produce the HelloWorld.class file.
2. **Go to** <TOMCAT_HOME>/webapps/axis2/WEB-INF.
3. **Create a directory named pojo** inside the **WEB-INF** directory.
4. Then **copy the HelloWorld.class into the pojo directory**, created in the previous step.
5. **Start Tomcat.**
6. **Go to** <http://localhost:8080/axis2/services/listServices>,

where you will be able to find a service named HelloWorld listed.

7. Now open your browser and type the following URL and see what you get.

<http://localhost:8080/axis2/services/HelloWorld/sayHello?name=Axis2>""

Now you should be able to see **Hello Axis2** in the browser.



We have successfully deployed our service more concretely; we **have invoked the service in a REST** (Representational State Transfer) **manner.**

Annotated POJO approach and POJO with packages

You can deploy both annotated Java classes and non annotated (plain) Java classes.

Annotations are a mechanism that provides metadata when using POJOs.

```
import javax.jws.WebService;  
import javax.jws.WebMethod;  
import javax.jws.WebParam;  
  
@WebService(targetNamespace = "http://sample.org/helloWorld", name =  
    "HelloWorld")  
public class HelloWorld {  
  
    @WebMethod(action = "urn:sayHello", operationName = "sayHello")  
    public String sayHello(@WebParam(partName = "name") String name)  
    {  
        return "Hello " + name;  
    }  
}
```

.....

When deploying a POJO as a single .class file, you cannot have the Java class declared inside a custom package.

If there are several classes that are required for the working of your POJO, and you cannot use the single class POJO approach any longer, then you need to follow the approach we are going to discuss here.

```
package book.sample
import javax.jws.WebService;
@WebService
public class AddressService {
    public Address getAddress(String name) {
        Address address = new Address();
        address.setStreet("Street");
        address.setNumber("Number 15");
        return address;
    }
}
```


You can see that we have annotated the AddressService class. The corresponding Address class is given here:

```
package book.sample
public class Address {
    private String street;
    private String number;
    public String getStreet() {
        return street;
    }
}
```

```
    public void setStreet(String street) {
        this.street = street;
    }
    public String getNumber() {
        return number;
    }
    public void setNumber(String number) {
        this.number = number;
    }
}
```

Now you have to compile the source code and create a Java ARchive file (JAR), that is, a .jar file from the .class files. There are multiple ways to create a JAR file.

Next, you will need to edit your axis2.xml to add a deployer to handle .jar files. You can do this by adding the following entry to the axis2.xml file in:



```
<TOMCAT_HOME>/webapps/axis2/WEB-INF/conf/axis2.xml
<!--.jar handler-->
<deployer extension=".jar" directory="pojo" class="org.apache.axis2.
deployment.POJODeployer"/>
```

The next step is to **drop the .jar file into**

```
<TOMCAT_HOME>/webapps/axis2/WEB-INF/pojo.A.jar
```

file will, more often than not, contain more than one .class file in it.

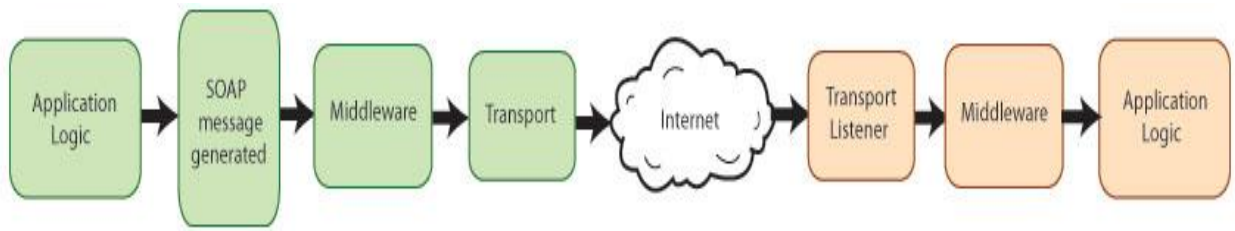
So there should be a way to identify the service class (or classes) in it. That is why we have annotated the POJO class with @WebService. Provided this annotation is applied, Axis2 will identify that class as the service implementation class and expose it as a web service.

To see what has happened (remember to restart Tomcat if it is already running), go to

<http://localhost:8080/axis2/services/listServices>, where you should

find a service called AddressService listed.

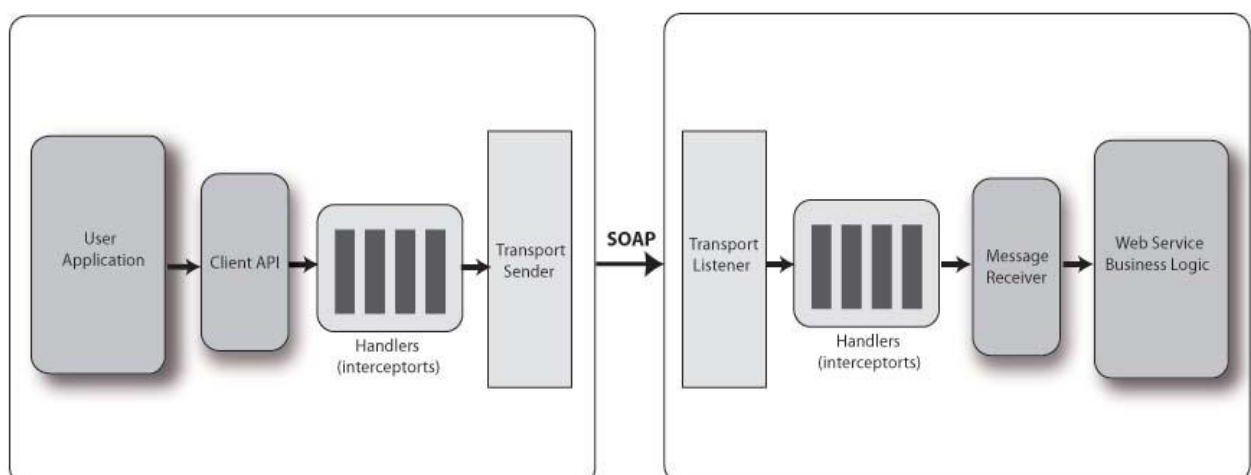
AXIS2 Architecture



The **sending application** creates the **original SOAP message**, an XML message that consists of headers and a body. (For more information on SOAP, see "[Introduction to Services](#)".) **If the system requires the use of WS* recommendations such as WS-Addressing or WS-Security, the message may undergo additional processing before it leaves the sender.** Once the message is ready, it is **sent via a particular transport such as HTTP, JMS, and so on.**

The message works its way over to the **receiver, which takes in the message via the transport listener.** (In other words, if the application doesn't have an HTTP listener running, it's not going to receive any HTTP messages.) Again, **if the message is part of a system that requires the use of WS-Security or other recommendations, it may need additional processing** for the purpose of checking credentials or decrypting sensitive information. Finally, a dispatcher determines the specific application (or other component, such as a Java method) for which the message was intended, **and sends it to that component.** That **component is part of an overall application designed to work with the data being sent back and forth.**

SOAP messages



The value of Web services is that the sender and receiver (each of which can be either the server or the client) don't even have to be on the same platform, much less running the same application. Assuming that Axis2 is running on both sides, the process looks like this:

- The sender creates the SOAP message.
- Axis "handlers" perform any necessary actions on that message such as encryption of WS-Security related messages.
- The transport sender sends the message.
- On the receiving end, the transport listener detects the message.
- The transport listener passes the message on to any handlers on the receiving side.
- Once the message has been processed in the "pre-dispatch" phase, it is handed off to the dispatchers, which pass it on to the appropriate application.

Code generation (from WSDL)

An important part of any SOAP processing framework is code generation based on WSDL

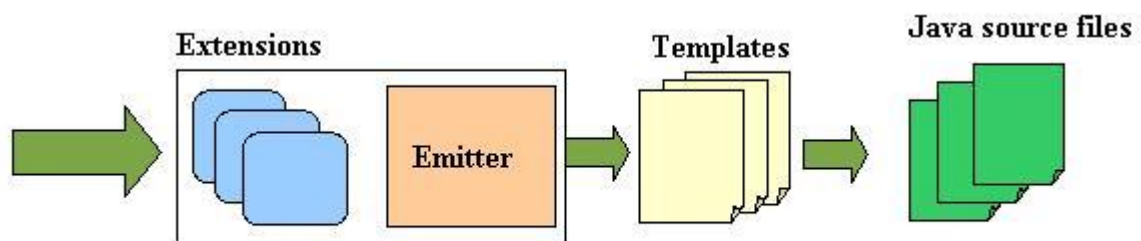


Figure1

The usual expectation of the code generation tool is the ability to process WSDL's and produce code to implement either the service or the client.


The code generation engine even though is the central component, does nothing other than simply calling the extensions one by one and then calling a component what is known as the **Emitter**. Emitter is the actual component that does the significant bit of work in the code generation process. Emitters are usually language dependent and hence one language has one emitter associated with it.

Flexibility of the tool comes at two levels.

1. **Extensions** Extensions can be used to implement either simple functions like filtering WSDL's or more complex functions such as processing schemas. In fact the databinding support for the code generator tool is incorporated by using extensions. However modifying the internal workings of an extension is a full job for a developer rather than a user.
2. **Templates** Templates provide a more user friendly way of customizing the actual text that would be generated. For a trivial usecase users can incorporate one of their own comments into the generated code by editing the templates. They can be used to even generate code for a completely different language by altering the template!

AXIOM

AXIOM stands for Axis2 Object Model and refers to the XML InfoSet model that was initially developed as part of Apache Axis2, but later it moved as a WS commons project so that projects other than Axis2 were able to benefit from it. XML

InfoSet refers to the information included inside the XML, and for programmatic manipulation it is convenient to have a representation of this XML InfoSet in a language-specific manner. 

For an object-oriented language, the obvious choice is a model made up of objects.

DOM and JDOM are two such XML models.

AXIOM is conceptually similar to such an XML model by its external behavior, but deep down it is very much different.

Topics:

- AXIOM architecture
- Pull parsing
- Working with AXIOM
- Advanced operations with AXIOM

AXIOM is not as complex as you may think.

Many APIs are very similar to JDOM APIs.

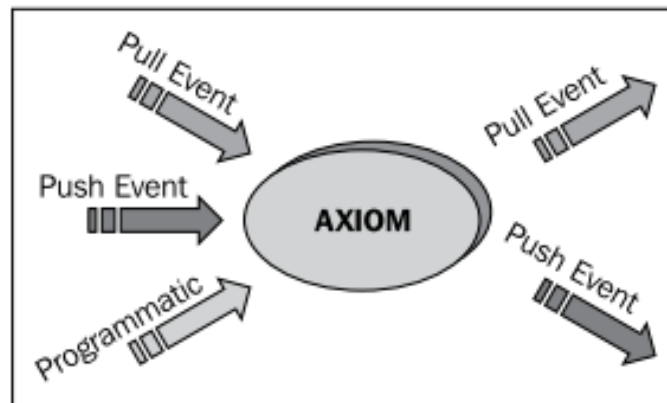
AXIOM has native support for binary data and MTOM, which is important for a SOAP processing framework such as Axis2.

AXIOM also has full support for XML processing and enhanced support for SOAP processing.

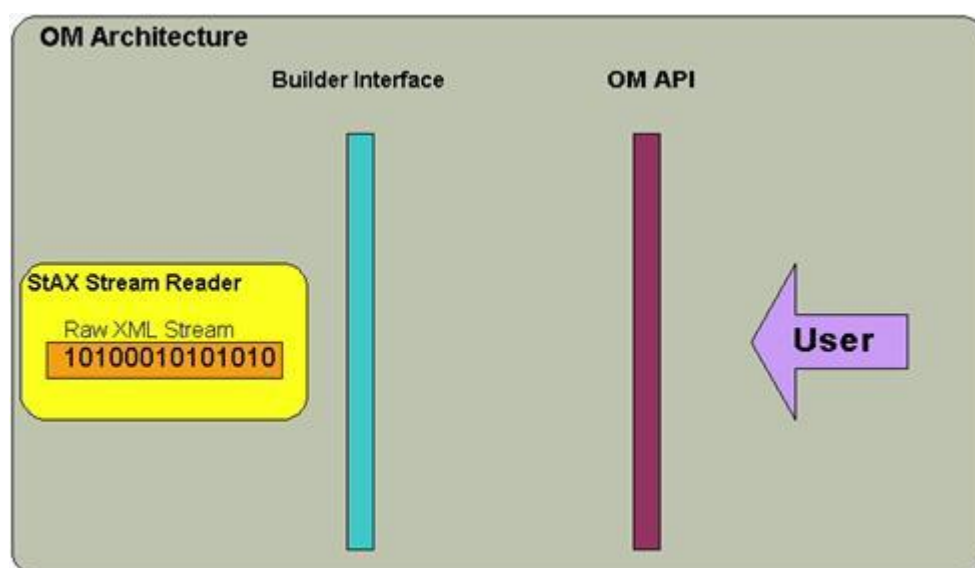
AXIOM is also known as Object Model (OM), and the OM Builder wraps the raw XML character stream through the StAX reader API.

Hence, the complexities of the pull event stream are transparent to the user.

You can create AXIOM (instance of object model) in three ways



- Axiom provides a notion of a factory and a builder to create objects.
- The factory helps to keep the code at the interface level and the implementations separately.
- Axiom is tightly bound to StAX API (API for pull parser).
- StAX-compliant reader should be created first with the desired input stream.
- One can then select one of the many builders available in AXIOM.



Axiom has OM builders (pure XML processing)
as well as SOAP builders (SOAP processing optimized for SOAP).

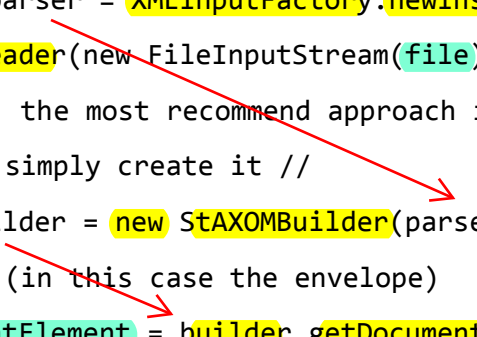
StAXOMBuilder will build a pure XML InfoSet-compliant object model,

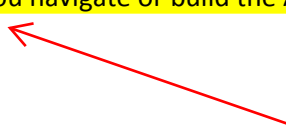
while the

SOAPModelBuilder returns SOAP-specific objects through its builder methods.

The following piece of code shows the correct method of creating an Axiom document from a file input stream (input stream can be file stream, socket stream or any other stream):

```
//First, create the parser
XMLStreamReader parser = XMLInputFactory.newInstance().
    createXMLStreamReader(new FileInputStream(file));
//create an OM builder, the most recommend approach is to use the //
//factory, but here we simply create it //
StAXOMBuilder builder = new StAXOMBuilder(parser);
//get the root element (in this case the envelope)
OMElement documentElement = builder.getDocumentElement();
```



1. When you want to read an input stream using Axiom, the first step is to create a parser with the input stream (in this case, we create a FileInputStream).
 2. Next, you need to create a builder by giving parser as an argument.
 3. The builder uses parser internally to process the input stream. Finally, you can get the root element;
 4. XML stream is still in the stream and no object tree is created at that time.
 5. The object tree is created only when you navigate or build the Axiom.
- 

AXIS2 Installation

Apache Axis2 Installation Guide

This document provides information on Axis2 distribution packages, system prerequisites and setting up environment variables and tools followed by detailed instructions on installation methods.

Send your feedback to: java-dev@axis.apache.org mailing list. (Subscription details are available on [Axis2 site](#).) Kindly prefix every email subject with [Axis2].

Contents

- [Axis2 Distributions](#)
- [System Requirements](#)
- [Installing Axis2 as a Standalone Server using the Standard Binary Distribution](#)
 - [Installing the Apache Axis2 Binary Distribution](#)
 - [Starting up Axis2 Standalone Server](#)
 - [Building the Axis2 WAR File Using the Standard Binary Distribution](#)
 - [Getting Familiar with Convenient Axis2 Scripts](#)
- [Installing Axis2 in a Servlet Container](#)
- [Uploading Services](#)
- [Advanced](#)
 - [Axis2 Source Distribution](#)
 - [Setting up the Environment and Tools](#)
 - [Building Axis2 Binaries and the WAR file Using the Source Distribution](#)

Axis2 Distributions

Axis2 is distributed in several convenient distribution packages and can be installed either as a standalone server or as part of a J2EE compliant servlet container. Axis2 is distributed under the Apache License, version 2.0. This Installation Guide will mainly focus on running Apache Axis2 using the Standard Binary Distribution.

[Download](#) distribution packages of the Apache Axis2 1.6.2 version (latest).

[Download](#) distribution packages of all versions of Apache Axis2.

The distribution packages provided are as follows:

1. Standard Binary Distribution



















This is the complete version of Axis2 and includes samples and convenient scripts as well.

[Download](#) Standard Binary Distribution

2. WAR (Web Archive) Distribution

This is the Web application of Axis2, which can be deployed in most of the servlet containers.

apache-tomcat-7.0.50\

Name	Name
 bin	 axis2
 conf	 docs
 lib	 examples
 logs	 host-manager
 temp	 manager
 webapps	 ROOT
 work	 axis2.war
 LICENSE	
 NOTICE	
 RELEASE-NOTES	
 RUNNING.txt	

[Download](#) WAR (Web Archive) Distribution

3. Documents Distribution

This contains all the documentation in one package. The package includes the xdocs and the Java API docs of this project.

[Download](#) Documents Distribution

4. Source Distribution

This contains the sources of Axis2 standard distribution, and is mainly for the benefit of advanced users. One can generate a binary distribution using the source by typing `$mvn -Drelease install`. You need to set up the Axis2 environment before running this command. Step by step details on how to create the binary distribution is available in the [Advanced](#) section. **System Requirements**

Java Development Kit (JDK)

1.5 or later (For instructions on setting up the JDK in different operating systems, visit <http://java.sun.com>)

Disk	Approximately 11 MB separately for standard binary distribution
Operating system	Tested on Windows XP, Linux, Mac OS X, Fedora core, Ubuntu, Gentoo
Build Tool- Apache Ant To run samples and to build WAR files from Axis2 binary distribution.	Version 1.6.5 or higher (download).
Build Tool- Apache Maven 2.x Required <i>only</i> for building Axis2 from Source Distribution	2.0.7 or higher in Maven 2.x series (download). Please download Maven 2.x version. Axis2 does not support Maven 1.x anymore.

Make sure that the above prerequisites are available for the Axis2 installation.

Installing Axis2 as a Standalone Server using the Standard Binary Distribution

This section provides you with the following information

1. Install Axis2 as a standalone server using the Standard Binary Distribution
2. Start up the Axis2 standalone server
3. Building the axis2.war file (using the Standard Binary Distribution) which is required to run Axis2 as part of a J2EE compliant servlet container
4. Running Axis2 convenient scripts

1. Download and Install the Apache Axis2 Binary Distribution

[Download](#) and install a Java Development Kit (JDK) release (version 1.5 or later). Install the JDK according to the instructions included with the release. Set an environment variable JAVA_HOME to the pathname of the directory into which you installed the JDK release.

Download and unpack the [Axis2 Standard Binary Distribution](#) into a convenient location so that the distribution resides in its own directory. Set an environment variable AXIS2_HOME to the pathname of the extracted directory of Axis2 (Eg: /opt/axis2-1.6.2). Linux users can alternatively run the setenv.sh file available in the AXIS2_HOME/bin directory to set the AXIS2_HOME environment variable to the Axis2 classpath.

2. Starting up Axis2 Standalone Server

The standalone Axis2 server can be started by executing the following commands:

```
%AXIS2_HOME%\bin\axis2server.bat (Windows)
```

```
$AXIS2_HOME/bin/axis2server.sh (Unix)
```

After startup, the default web services included with Axis2 will be available by visiting <http://localhost:8080/axis2/services/>

3. Building the Axis2 Web Application (axis2.war) Using Standard Binary Distribution

[Download](#) and install Apache Ant (version 1.6.5 or later). Install Apache Ant according to the instructions included with the Ant release.

Locate the Ant build file (build.xml) inside the webapp directory, which resides in your Axis2 home directory (i.e.: - \$AXIS2_HOME/webapp). Run the Ant build by executing "ant create.war" inside the \$AXIS2_HOME/webapps folder. You can find the generated axis2.war inside the \$AXIS2_HOME/dist directory. All the services and modules that are present in the \$AXIS2_HOME/repository will be packed into the created axis2.war together with the Axis2 configuration found at \$AXIS2_HOME/conf/axis2.xml.

Read [Installing Axis2 in a Servlet Container](#) to find out how to deploy the Axis2 Web application in a servlet container.

4. Getting Familiar with the Convenient Axis2 Scripts

It is advised to add the \$AXIS2_HOME/bin to the PATH, so that you'll be able to run the following scripts from anywhere.

Script Name	Description
axis2.{bat sh}	<p>You can use this script to run web service clients written using Axis2. This script calls the "java" command after adding the classpath for Axis2 dependent libraries (*.jar files present in your \$AXIS2_HOME/lib), setting the Axis2 repository location (\$AXIS2_HOME/repository) and setting the Axis2 configuration file location(\$AXIS2_HOME/conf/axis2.xml) for you. With this you can be relieved from setting all the above Axis2 specific parameters.</p> <p><i>Usage : axis2.{sh.bat} [-options] class [args...]</i></p>
axis2server.{sh bat}	<p>This script will start a standalone Axis2 server using the \$AXIS2_HOME/repository as the Axis2 repository and the \$AXIS2_HOME/conf/axis2.xml as the Axis2 configuration file. This will start all the transport listeners listed in the</p>

	<p>AXIS2_HOME/conf/axis2.xml.</p> <p>For example, if you want to deploy a service using a standalone Axis2 server, then copy your service archive to the AXIS2_HOME/repository/services directory. Next, go to the "Transport Ins" section of the AXIS2_HOME/conf/axis2.xml and configure the transport receivers (simpleHttpServer in port 8080 is listed by default). Then invoke this script.</p> <p>The server can be started in debug mode by adding the <code>-xdebug</code> option to the command line. A remote debugger can then be attached by connecting to port 8000.</p>
wSDL2Java.{bat sh}	<p>This script generates Java code according to a given WSDL file to handle Web service invocations (client-side stubs). This script also has the ability to generate web service skeletons according to the given WSDL.</p> <p><i>Usage: wSDL2Java.{sh bat} [OPTION]... -uri <Location of WSDL></i></p> <p>e.g., <code>wSDL2Java.sh -uri ../wSDL/Axis2Sample.wsdl</code></p> <p>A more detailed reference about this script can be found here</p>
Java2WSDL.{bat sh}	<p>This script generates the appropriate WSDL file for a given Java class.</p> <p><i>Usage: Java2WSDL.{sh bat} [OPTION]... -cn <fully qualified class name></i></p> <p>e.g., <code>Java2WSDL.sh -cn ../samples/test/searchTool.Search</code></p> <p>A more detailed reference about this script can be found here</p>

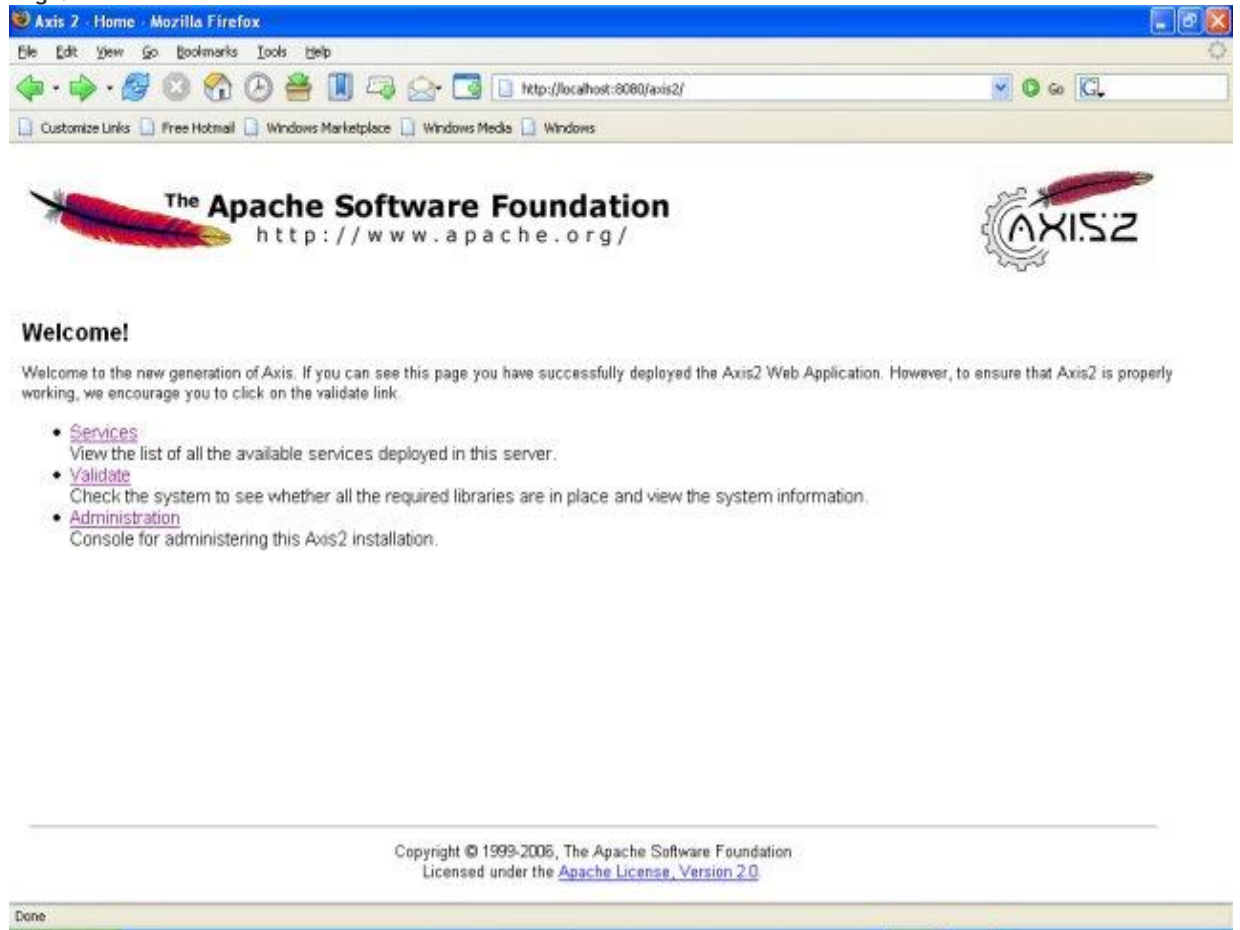
Installing Axis2 in a Servlet Container

Whichever the distribution, installing Axis2 in a J2EE compliant servlet container is as follows:

1. Build the Axis2 WAR file using the Axis2 [Standard Binary Distribution](#). (Alternatively you can [download](#) the axis2.war file or you can build axis2.war using the [Source Distribution](#).)

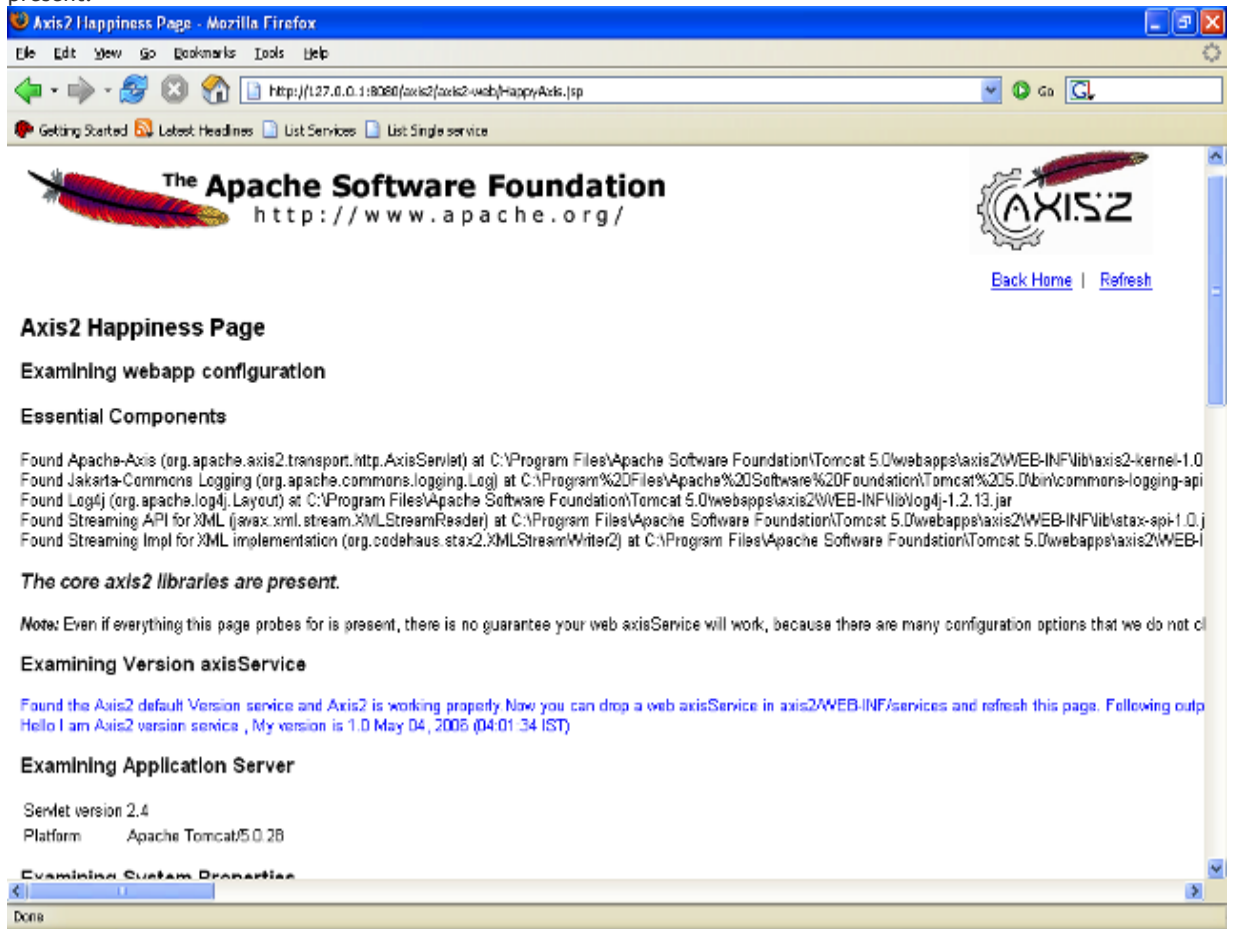
- Drop the WAR file in the webapps folder of the servlet container. Most servlet containers will automatically install the WAR file. (Some servlet containers may require a restart in order to capture the new web application. Refer to your servlet container documentation for more information.)
- Once the WAR is successfully installed, test it by pointing the web browser to the `http://<host>:<port>/axis2`. It should produce the following page which is the **Axis2 Web Application Home**

Page.



- Use the link "Validate" to ensure that everything is running correctly. If the validation fails then the WAR has failed to install properly or some essential jars are missing. In such a situation, refer to the documentation of the particular servlet container to find the problem. The following page shows a successful validation. Note the statement that indicates the core Axis2 libraries are

present.



Note: For any Application server specific installation information please refer to the [Application Server Specific Configuration Guide](#).

Uploading Services

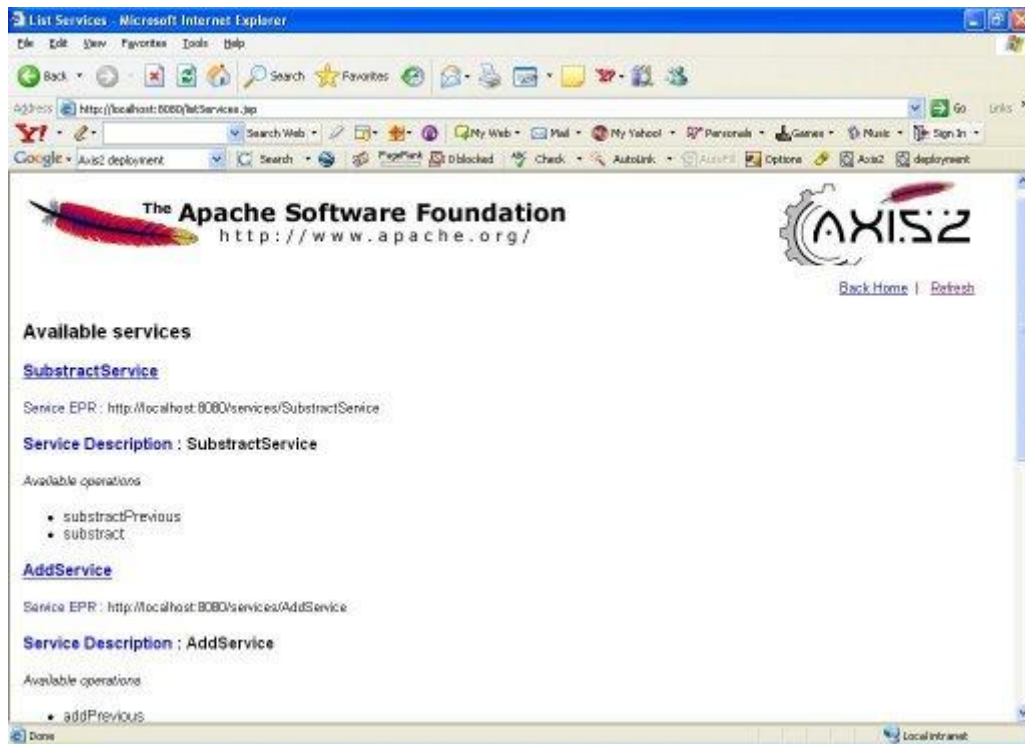
The Axis2 Web application also provides an interface to upload services. Once a service archive file is created according to the service specification as described in the Advanced User's Guide, that .aar file can be uploaded using the upload page.



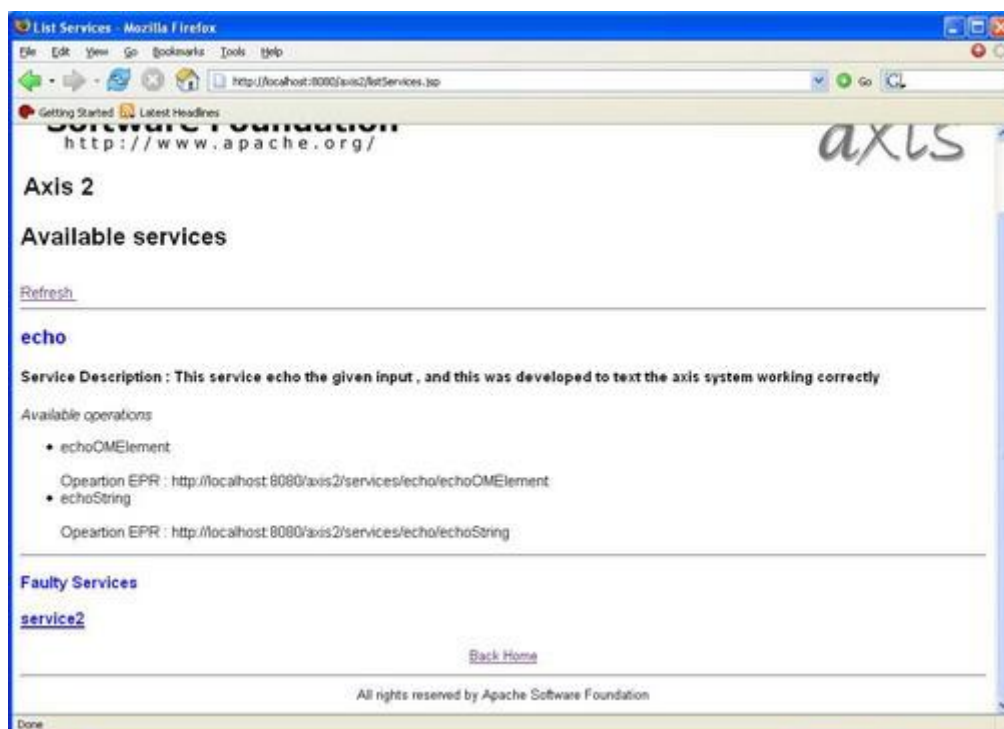
The uploaded .aar files will be stored in the default service directory. For Axis2, this will be the <webapps>/axis2/WEB-INF/services directory. Once a service is uploaded, it will be installed instantly.

Since Axis2 supports **hot deployment**, you can drop the service archive directly through the file system to the above mentioned services directory. It will also cause the service to be automatically installed without the container being restarted.

Use the 'Services' link on the Web Application home page to check the successful installation of a service. The services and the operations of successfully installed services will be displayed on the available services page.



If the service has deployment time errors it will list those services as faulty services. If you click on the link, you will see the deployment fault error messages.



Deployment time error message



Axis2 Administration is all about configuring Axis2 at the run time and the configuration will be transient. More descriptions are available in the [Axis2 Web Administration Guide](#)

Advanced

Axis2 Source Distribution

By using the Source Distribution, both binary files (which can be downloaded as the [Standard Binary Distribution](#)) and the axis2.war file (which can be downloaded as the [WAR distribution](#)) can be built using Maven commands.

Required jar files do not come with the distribution and they will also have to be built by running the maven command. Before we go any further, it is necessary to install [Maven2](#) and set up its environment, as explained below.

Setting Up the Environment and Tools

Maven

The Axis2 build is based on [Maven2](#). Hence the only prerequisite to build Axis2 from the source distribution is to have Maven installed. Extensive instruction guides are available at the Maven site. This guide however contains the easiest path for quick environment setting. Advanced users who wish to know more about Maven can visit [this site](#).

- MS Windows
 1. Download and run the Windows installer package for Maven.
 2. Set the 'Environment Variables' (create system variable MAVEN_HOME and edit path. eg: "C:\Program Files\Apache Software Foundation\maven-2.0.7"; path %MAVEN_HOME%\bin)
 3. Make sure that the system variable JAVA_HOME is set to the location of your JDK, eg. C:\Program Files\Java\jdk1.5.0_11
 4. Run mvn -v or mvn -version to verify that it is correctly installed.

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text:

```
C:\>mvn -v
Maven version: 2.0.7
Java version: 1.5.0_06
OS name: "windows xp" version: "5.1" arch: "x86"
C:\>
```

- Unix based OS (Linux etc)

The tar ball or the zip archive is the best option. Once the archive is downloaded expand it to a directory of choice and set the environment variable MAVEN_HOME and add MAVEN_HOME/bin to the path as well. [More instructions](#) for installing Maven in Unix based operating systems.

Once Maven is properly installed, you can start building Axis2.

[Maven commands that are frequently used](#) in Axis2 are listed on the [FAQs](#) page.

Building Binaries and the WAR File Using the Source Distribution

The Source Distribution is available as a zipped archive. All the necessary build scripts are included with the source distribution. Once the source archive is expanded into a directory of choice, moving to the particular directory and running `mvn install` command will build the Axis2 jar file.

Once the command completes, the binaries (jar files in this case) can be found at a newly created "target" directory.

Note: For the first Maven build (if the maven repository is not built first) it will take a while since the required jars need to be downloaded. However, this is a once only process and will not affect any successive builds.

The default maven build will generate the war under `modules/webapp/target` directory