Enabling a third-party JAX-WS application in WebSphere Application Server V7

Nikhil Thaker
Senior Software Engineer
Web Services Architect
IBM, Austin, TX

January, 2010

This article describes how you can use a third-party Java™ API for XML Web services (JAX-WS) engine in WebSphere Application Server V7. It describes how to deploy an application using an external third-party JAX-WS engine, as well as limitations and potential issues.

Introduction

Some WebSphere Application Server (hereafter Application Server) users would like to leverage a third-party JAX-WS engine like Axis2 or Apache CXF by having these runtimes embedded in an application EAR file. There are several reasons you might want to do this, such as to use features specific to the third-party runtimes, to reuse code, or to achieve application consistency in a multi-vendor or multi-version environment.

When such a third-party application is deployed on Application Server, the Application Server JAX-WS Web services engine must be disabled so it doesn't conflict with the third-party implementation. Starting with Fixpack 7, configuration options were added to enable this on a per-application or per-server basis.

Use of third-party JAX-WS runtimes has limitations and requires certain configuration changes. In some cases, it may also require manual intervention to resolve issues that occur during deployment and runtime. These limitations and issues vary based on the external runtime.

This article demonstrates how to use an application EAR that has Axis2 or Apache CXF embedded in it.

Note: WebSphere does not claim support for any of the third-party JAX-WS runtimes. IBM ensures successful deployment of such applications only. See the Tech Note for more information.

Limitations when using a third-party JAX-WS runtime

Following are the observed limitations when deploying an application EAR with a third-party JAX-WS on Application Server:

- The WebSphere runtime restricts the usage of application modules that use both IBM's JAX-WS implementation and an external JAX-WS implementation in the same application EAR. You can use the IBM JAX-WS implementation or an external implementation in a single application EAR, but not both. This limitation ensures that the Application Server runtime does not conflict with the external third-party JAX-WS implementation.
- Most of the external third-party JAX-WS runtimes include various JAR libraries, and some of those are already loaded by the Application Server runtime, causing conflicts. If you try to deploy an application with an external JAX-WS implementation and there is a conflicting JAR file in the application, you'll have to remove the conflicting JARs. Some of these conflicts are described in the sections Troubleshoot Axis2 deployment and Troubleshoot CXF application deployment. Following are the jars known to conflict with the Application Server runtime:
    - saaj-api.jar
    - saaj-impl.jar
    - xalan.jar
    - xercesImpl.jar

    - xml-apis.jar
    - jaxb-api.jar
    - jaxb-impl.jar

- Once an application with third-party JAX-WS runtime is deployed on Application Server, it won't be recognized as a service client or provider and therefore users won't be able to attach application level policy sets. Users will have to rely on external runtimes to support Quality of Service (QoS). Following is a list of Application Server features that users won't be able to use when they choose to deploy an application that uses third-party JAX-WS implementations:

    - WS-Security, WS-RM, WS-Transaction, policy sets
    - WSDM
    - JNDI lookup to retrieve JAX-WS service or port instance
    - Fine-grained administration of services and endpoints

Required configuration changes

The following configuration changes are required to use a third-party JAX-WS runtime.

- In order to use an external JAX-WS runtime, you'll need to change the class loader Order policy in Application Server at the module level. For V7.0, this policy needs to be set to "Classes loaded with local class loader first (parent last)".

  Changing the class loader policy to parent last ensures that the external third-party JAX-WS runtime and its dependent JAR files are first in the class loader search path

- In order to deploy and run applications with third-party JAX-WS runtimes, you need to configure Application Server to turn off Web services annotation scanning. Annotation scanning can be turned off at application level or at server level. To do this, complete the steps in Disabling IBM JAX-WS implementation.

Building and deploying an Axis2 Web services application

This section describes the steps to build, deploy, and run an Axis2 Web services application, as well as how to troubleshoot potential problems.

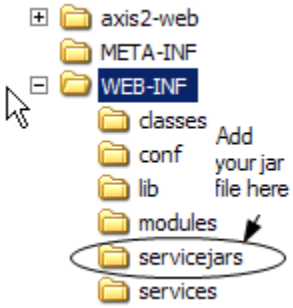
Build the Axis2 application

To build an Axis2 application, do the following:

1. Download the latest release of the Axis2 runtime.
2. The Axis2 runtime is packaged as a WAR called axis2.war, which can be imported into Rational Application Developer (hereafter Application Developer) as a dynam-

ic Web project. When you import the axis2.war into Application Developer, you'll notice that the WebContent/WEB-INF/lib has all the dependent jars that you'll need to build an Axis2 application.

3. Create a new Java project in Application Developer. Update the build path to add a Web application library and point it to the imported axis2.war.
4. Follow the instructions in the Axis2 documentation to build a JAX-WS Web service application.
5. Package your service as a JAR file, then create a folder called WebContent/WEB-INF/servicejars under axis2.war, as shown below, and place the packaged jar in the new folder.



6. Follow the instructions in Deploy the Axis2 WAR to deploy axis2.war.

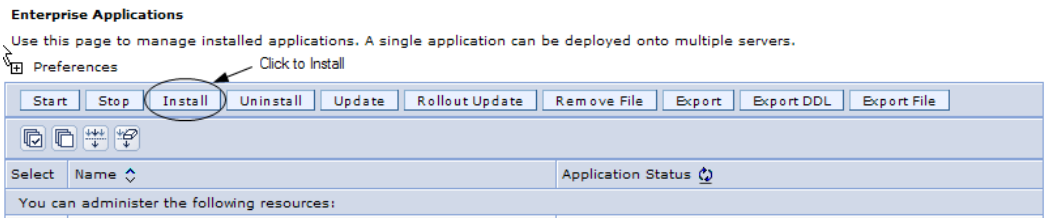
Deploy the Axis2 WAR

Complete the following steps to deploy an Axis2 runtime on Application Server:

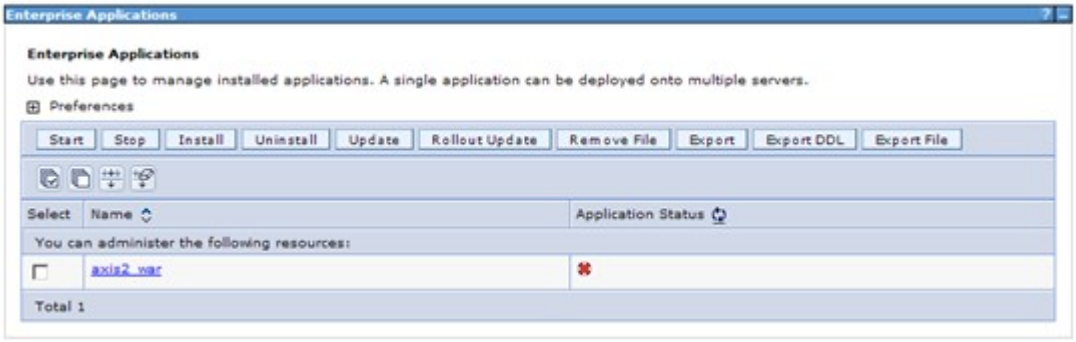
1. Turn off Web services annotation scanning as described in Disabling IBM JAX-WS implementation.

2. Open the Application Server administrative console and select **Applications => Application Types => WebSphere enterprise applications**, as shown below.

3. Click **Install** as shown below.

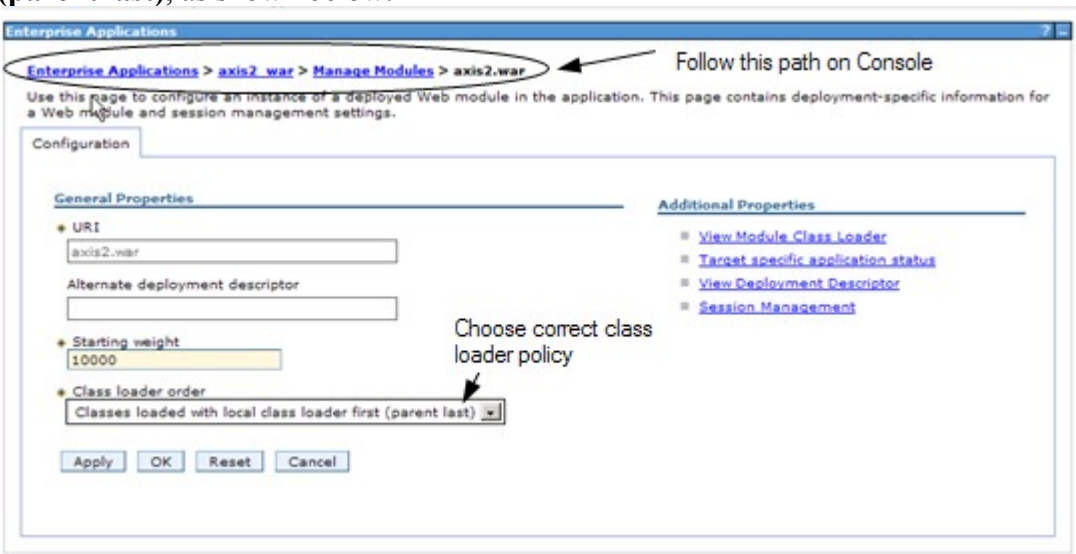


4. Follow the wizard and install the axis2.war file that was downloaded from the Apache Web site.



5. Ensure that parent last class loader policy is used. This class loader policy ensures that all the Web application libraries will be installed first in the classpath, thereby ensuring that the CXF Web services provider is used instead of the IBM Web services provider. To do this, go to the General Properties of the module file and change the **Class Loader order** to **Classes loaded with local class loader first**

(parent last), as shown below.



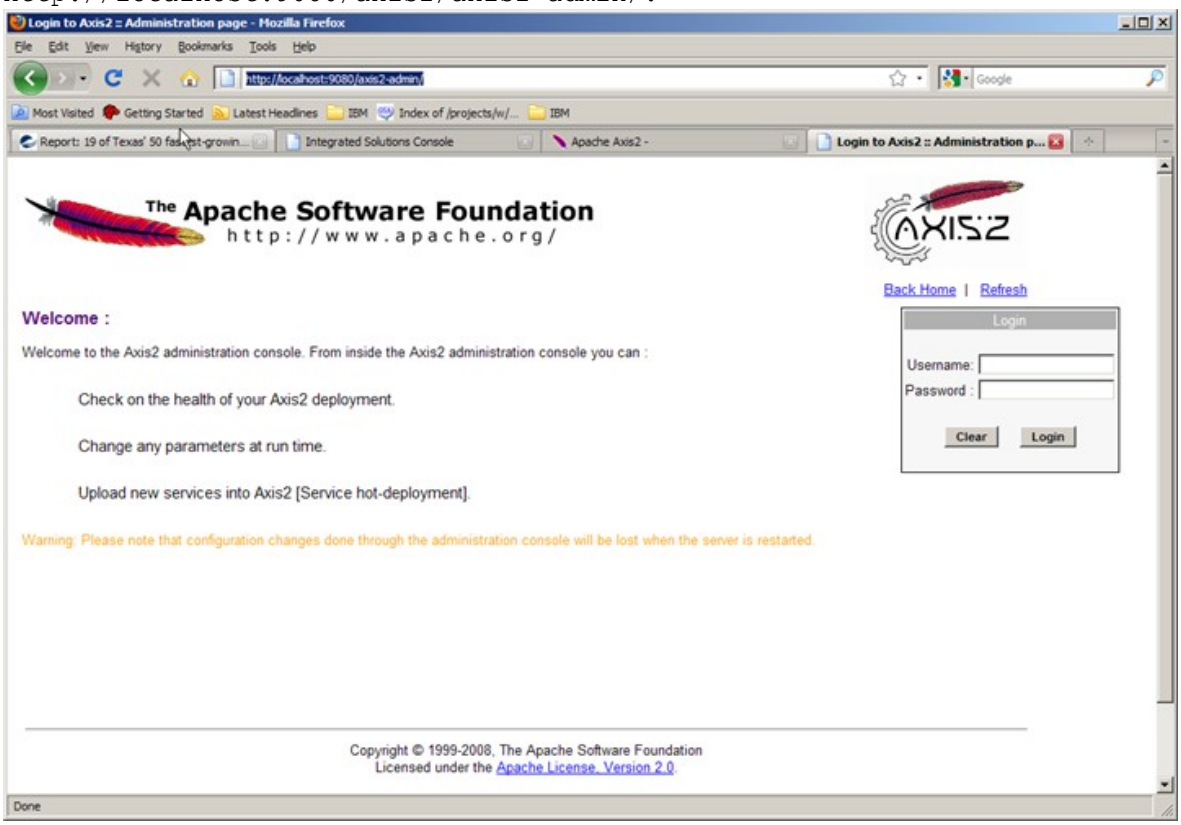
6. Click **Apply** and then save your changes.
7. Restart the server and make sure that the application has deployed and started successfully.

Deploy the application using Axis2 administration

Complete the following steps to deploy an Axis2 Web services application using the Axis administration screen in your browser:

1. First complete the steps to deploy the Axis2 runtime on WebSphere, as described in Deploy the Axis2 WAR.
2. Point your Web browser to `http://localhost:9080/axis2-admin/`, as shown below. Change the default port number of 9080 as needed. Note that in this example, the context-root for axis2.war was "/" but you can choose to use axis2 as the context-root, in which case the URL would be

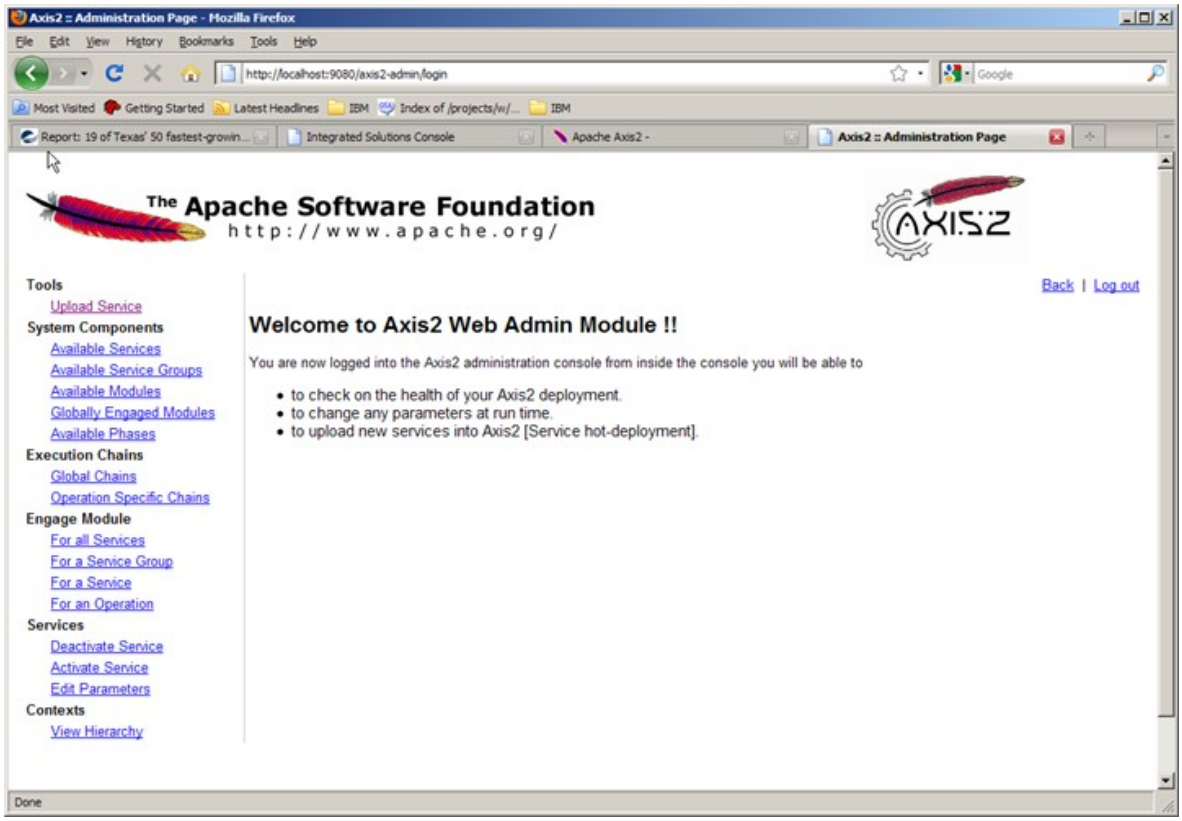
`http://localhost:9080/axis2/axis2-admin/.`



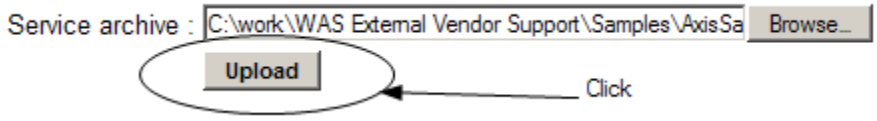
3. Enter the username and password for the administrator screen. The user name and password are located in the axis2.xml file that was deployed with the axis2.war.

```
<parameter name="userName" locked="false">admin</parameter>
<parameter name="password" locked="false">axis2</parameter>
```

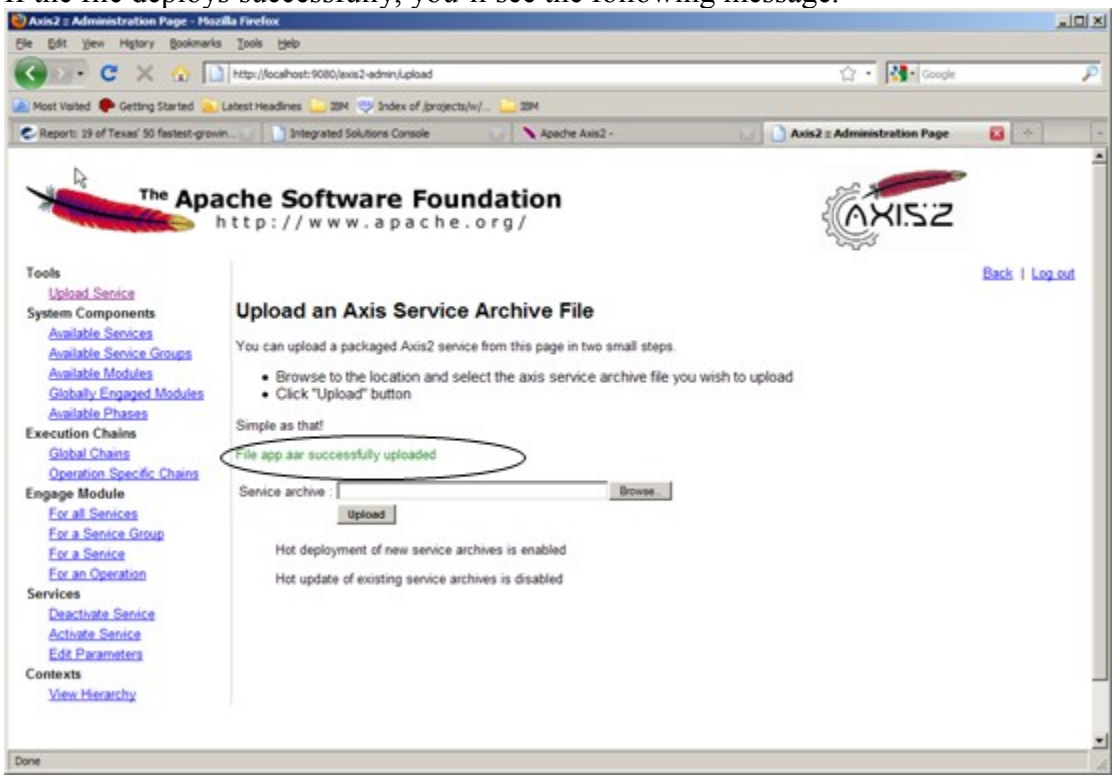
4. Select **Upload Service**, as shown below.



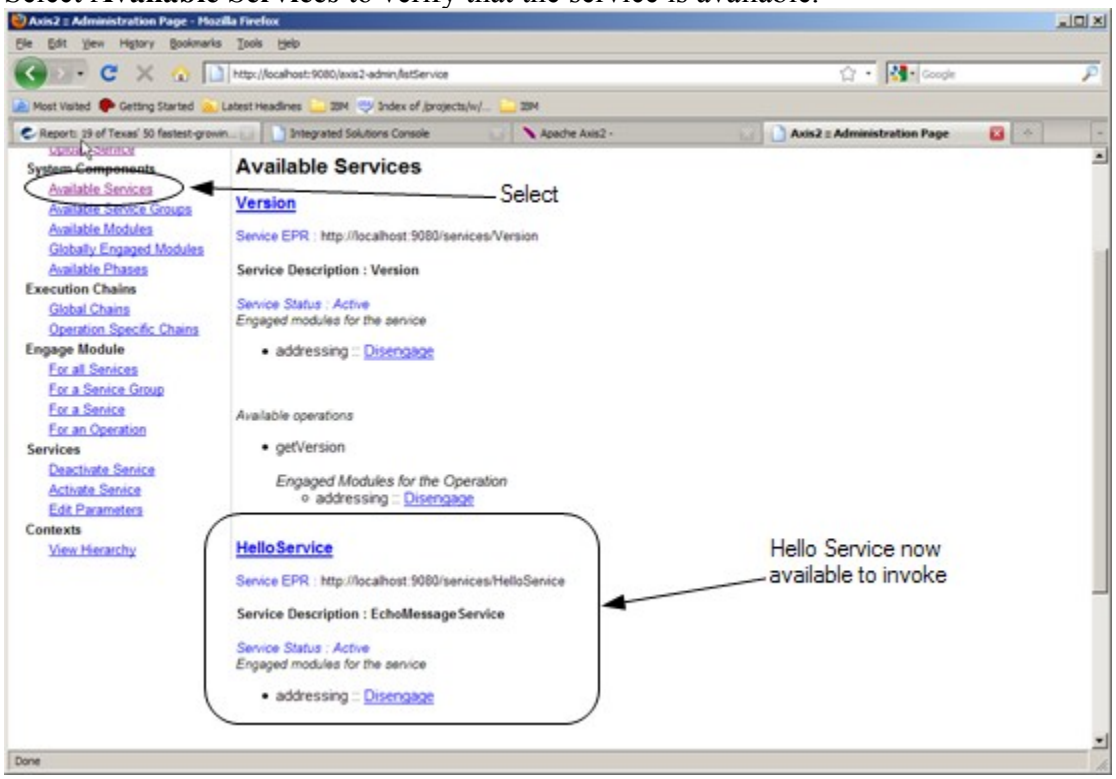
5. Browse to the Axis2 AAR file you want to deploy and click upload.



6. If the file deploys successfully, you'll see the following message.



7. Select **Available Services** to verify that the service is available.



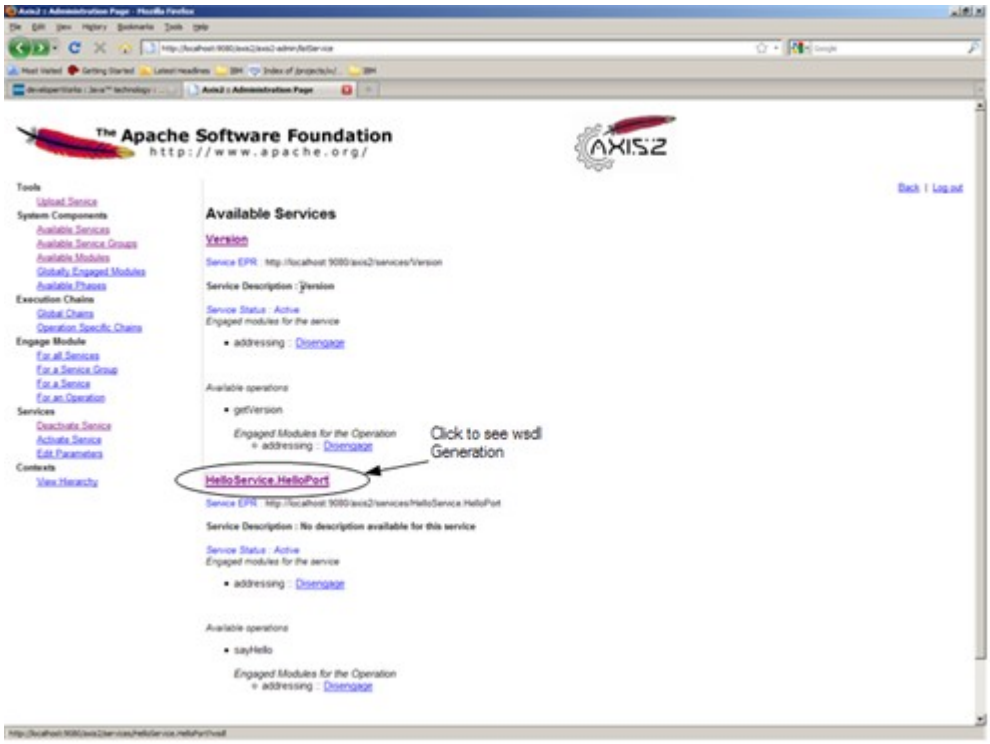
8. Execute the Axis2 JAX-WS client to invoke the service.

Troubleshoot Axis2 deployment

While testing Axis2 JAR file deployment and execution, we identified the following issues.

Conflicting axis2-saaj-<version>.jar

After following the steps to deploy an Axis application on Application Server V7.0.0.7, we encountered an error when trying to generate WSDL by clicking **HelloService HelloPort** on the Axis2 administration screen as shown below.



After the failed attempt to generate the WSDL, the following errors were seen in the WebSphere trace.log:

```
Caused by: java.lang.Error: javax.xml.soap.SOAPException: Unable to create SAAJ meta-factoryorg.apache.axis2.saaj.SAAJMetaFactoryImpl incompatible with
javax.xml.soap.SAAJMetaFactory
    at com.sun.xml.internal.ws.api.SOAPVersion.<init>(SOAPVersion.java:166)
    at com.sun.xml.internal.ws.api.SOAPVersion.<clinit>(SOAPVersion.java:68)
    at java.lang.J9VMInternals.initializeImpl(Native Method)
    at java.lang.J9VMInternals.initialize(J9VMInternals.java:200)
    at com.sun.xml.internal.ws.api.BindingID.<clinit>(BindingID.java:304)
    at java.lang.J9VMInternals.initializeImpl(Native Method)
    at java.lang.J9VMInternals.initialize(J9VMInternals.java:200)
    at com.sun.tools.internal.ws.wscompile.WsgenOptions.validateBinding(WsgenOptions.java:240)
    at com.sun.tools.internal.ws.wscompile.WsgenOptions.validateEndpointClass(WsgenOptions.java:235)
    at com.sun.tools.internal.ws.wscompile.WsgenOptions.validate(WsgenOptions.java:202)
    at com.sun.tools.internal.ws.wscompile.WsgenTool.run(WsgenTool.java:123)
    at com.sun.tools.internal.ws.util.WSToolsObjectFactoryImpl.wsgen(WSToolsObjectFactoryImpl.java:47)
    at com.sun.tools.internal.ws.spi.WSToolsObjectFactory.wsgen(WSToolsObjectFactory.java:93)
    ... 32 more
Caused by: javax.xml.soap.SOAPException: Unable to create SAAJ meta-factoryorg.apache.axis2.saaj.SAAJMetaFactoryImpl incompatible with
javax.xml.soap.SAAJMetaFactory
    at javax.xml.soap.SAAJMetaFactory.getInstance(SAAJMetaFactory.java:81)
    at javax.xml.soap.MessageFactory.newInstance(MessageFactory.java:144)
    at com.sun.xml.internal.ws.api.SOAPVersion.<init>(SOAPVersion.java:163)
    ... 44 more
```

It's evident that the "org.apache.axis2.saaj.SAAJMetaFactoryImpl incompatible with javax.xml.soap.SAAJMetaFactory" error message is the result of Axis2 SAAJ problems.

If you don't plan to use Axis2 SAAJ in your application, we recommend that you remove the axis2-saaj-<version>.jar and axis2-saaj-api-<version>.jar files from WEB-INF/lib folder of the axis2.war file and redeploy the service.

If you plan to use Axis2 SAAJ in your application, you'll need to resolve this issue by setting up class loading in Application Server so the `SAAJMetaFactoryImpl` and `SAAJ-MetaFatory` interfaces are loaded from a single class loader. For more information on how to debug the problem, refer to WebSphere Application Server Class Loader Problem Determination.

Exception when using javax.xml.transform.TransformerFactory

During a `?wsdl` call from the client we encountered the following error in WebSphere Application server trace.log.

```
[7/9/09 16:45:23:560 EDT] 00000011 webapp        E com.ibm.ws.webcontainer.webapp.WebApp logServletError SRVE0293E: [Servlet Error]-[AxisServlet]:
java.lang.ClassCastException: com.ibm.xtq.xslt.jaxp.compiler.TransformerFactoryImpl incompatible with javax.xml.transform.TransformerFactory
    at javax.xml.transform.TransformerFactory.newInstance(Unknown Source)
    at org.apache.ws.commons.schema.XmlSchema.serialize_internal(XmlSchema.java:453)
    at org.apache.ws.commons.schema.XmlSchema.write(XmlSchema.java:426)
    at org.apache.axis2.description.AxisService.printXSD(AxisService.java:1232)
    at org.apache.axis2.transport.http.ListingAgent.processListService(ListingAgent.java:294)
    at org.apache.axis2.transport.http.AxisServlet.doGet(AxisServlet.java:242)
```

The error "java.lang.ClassCastException: com.ibm.xtq.xslt.jaxp.compiler.TransformerFactoryImpl incompatible with javax.xml.transform.TransformerFactory" occurs because Application Server does not use `javax.xml.transform.TransformerFactory` defined in the `META-INF/services` file of `xalan-<version>.jar.`

You can resolve this issue by creating a jaxp.properties file in <WAS_HOME>\as\java\jre\lib and adding the following properties, as shown below.

```
#
#javax.xml.transform.TransformerFactory=com.ibm.xtq.xslt.jaxp.compiler.TransformerFactoryImpl
javax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl
#javax.xml.xpath.XPathFactory=org.apache.xpath.jaxp.XPathFactoryImpl
#javax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
#javax.xml.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
#javax.xml.validation.SchemaFactory:http\://www.w3.org/2001/XMLSchema=org.apache.xerces.jaxp.validation.XMLSchemaFactory
#javax.xml.datatype.DatatypeFactory=org.apache.xerces.jaxp.datatype.DatatypeFactoryImpl
```

Adding the `javax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl` property ensures that proper transform factory is used by the Axis2 application.

Commons-logging implementation for the Axis2 application doesn't work

In some cases, we found that the commons-logging implementation didn't work for applications deployed in the servicejars folder.

WebSphere ships the file commons-logging.properties, which is located in the class loader path. Therefore, this is the properties file that WebSphere picks up by default. It also ships with the log implementation set to JDK14 logging (`org.apache.commons.logging.Log=org.apache.commons.logging.impl.JDK14Logger`), causing problems when you try to implement commons-logging in Axis2 applications.

To ensure that commons logging works:

- Ensure that commons-logging.properties and log4j.properties are located in Axis2.war WEB-INF/Classes folder.
- Ensure your log4j.properties file is configured correctly.
- Modify Commons-logging.properties to have `priority=1`. The priority flag was introduced in commons-logging 1.1 to allow ordering based on the priority. This flag ensures that applications commons-logging takes precedence over WebSphere's commons-logging.properties file.

After making these changes, commons-logging should work from Axis2 applications.

Build and deploy a CXF Web services application

This section describes the steps to build, deploy, and run a CXF Web services application, as well as how to troubleshoot potential problems.

Build the CXF application

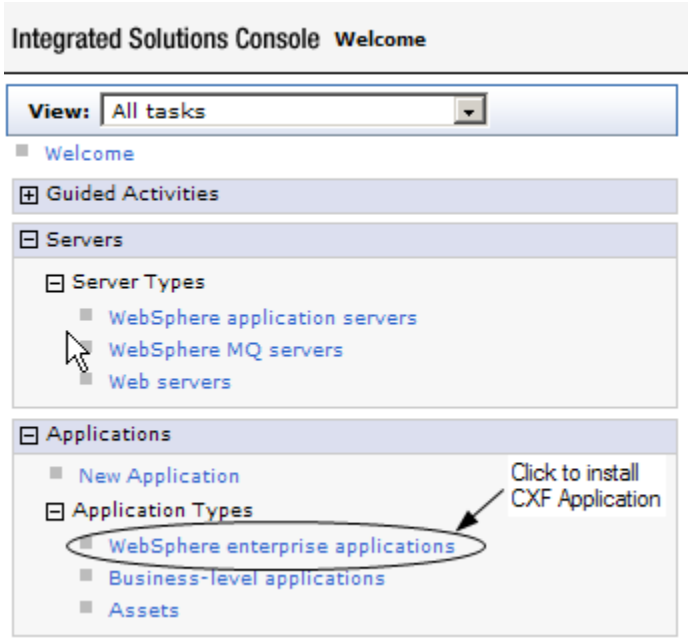
To build a CXF application, do the following:

1. Download CXF version 2.2.1.
2. Use the Application Developer **Create a Dynamic Web Project** function and copy all the dependent files from <apache-cxf-version>/lib into the Web module's WebContent/WEB-INF/lib folder in your Application Developer environment. You may choose not to copy all the library files from the CXF installation. You can get a list of the dependent files from the Apache CXF documentation.
3. Build the Web service application in the above Web module. For more information on how to create a Web service using the CXF runtime, refer to the CXF documentation.
4. Once you've built your application, export the EAR file from Application Developer and follow the steps in the next section to deploy the application.

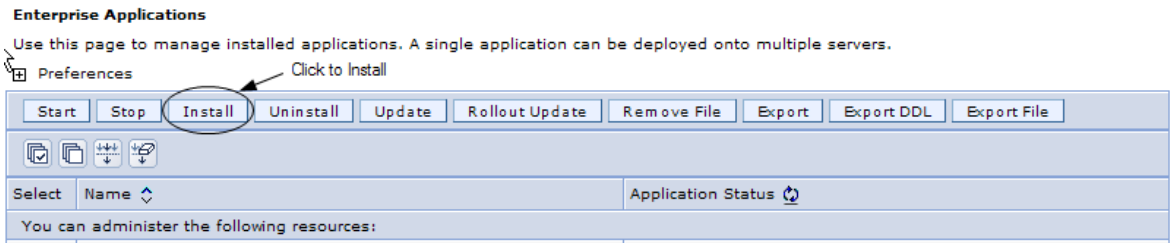
Deploy the CXF application

To deploy a CXF application, do the following:

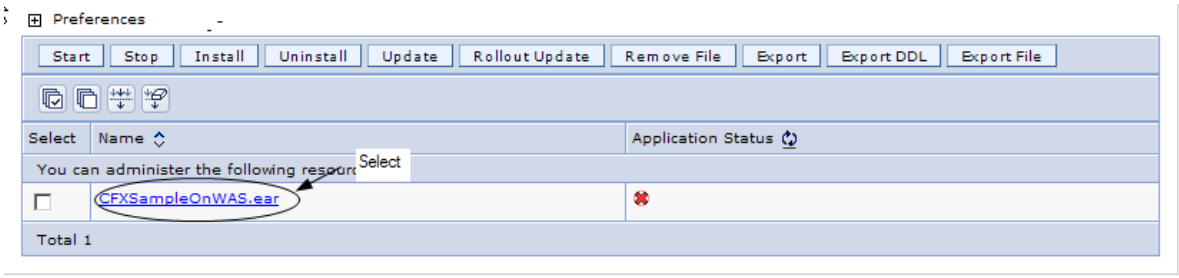
1. Turn off Web services annotation scanning as described in Disabling the IBM JAX-WS implementation.
2. Open the Application Server administrative console and select **Applications => Application Types => WebSphere enterprise applications**, as shown below.



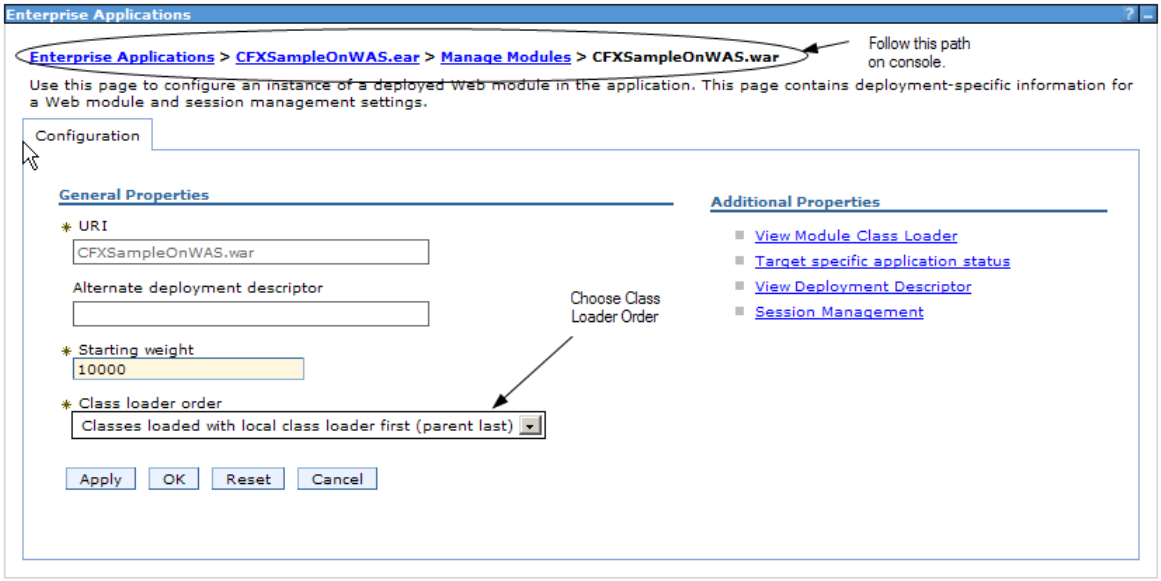
3. Click **Install** as shown below.



4. Use the wizard to install the exported EAR file from Application Developer. After you've installed the file, you'll see the following:



5. Ensure that parent last class loader policy is used. This class loader policy ensures that all the Web application libraries will be installed first in the classpath, thereby ensuring that the CXF Web services provider is used instead of the IBM Web services provider. To do this, go to the General Properties of the module file and change the **Class loader order** to **Classes loaded with local class loader first (parent last)**, as shown below.



6. Click **Apply** and save your changes.
7. Restart the server and make sure that the application has deployed and started successfully.

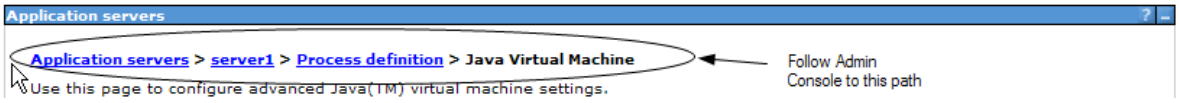
Troubleshoot CXF application deployment

The geronimo-servlet_2.5_spec-1.2.jar provided with CXF V2.2.2 conflicts with the Application Server servlet implementation. This shouldn't occur if you are using CXF 2.2.1 as instructed in Build an Axis2 application. To fix this problem, remove the conflicting jar from the WEB-INF/lib folder of the packaged WAR file.

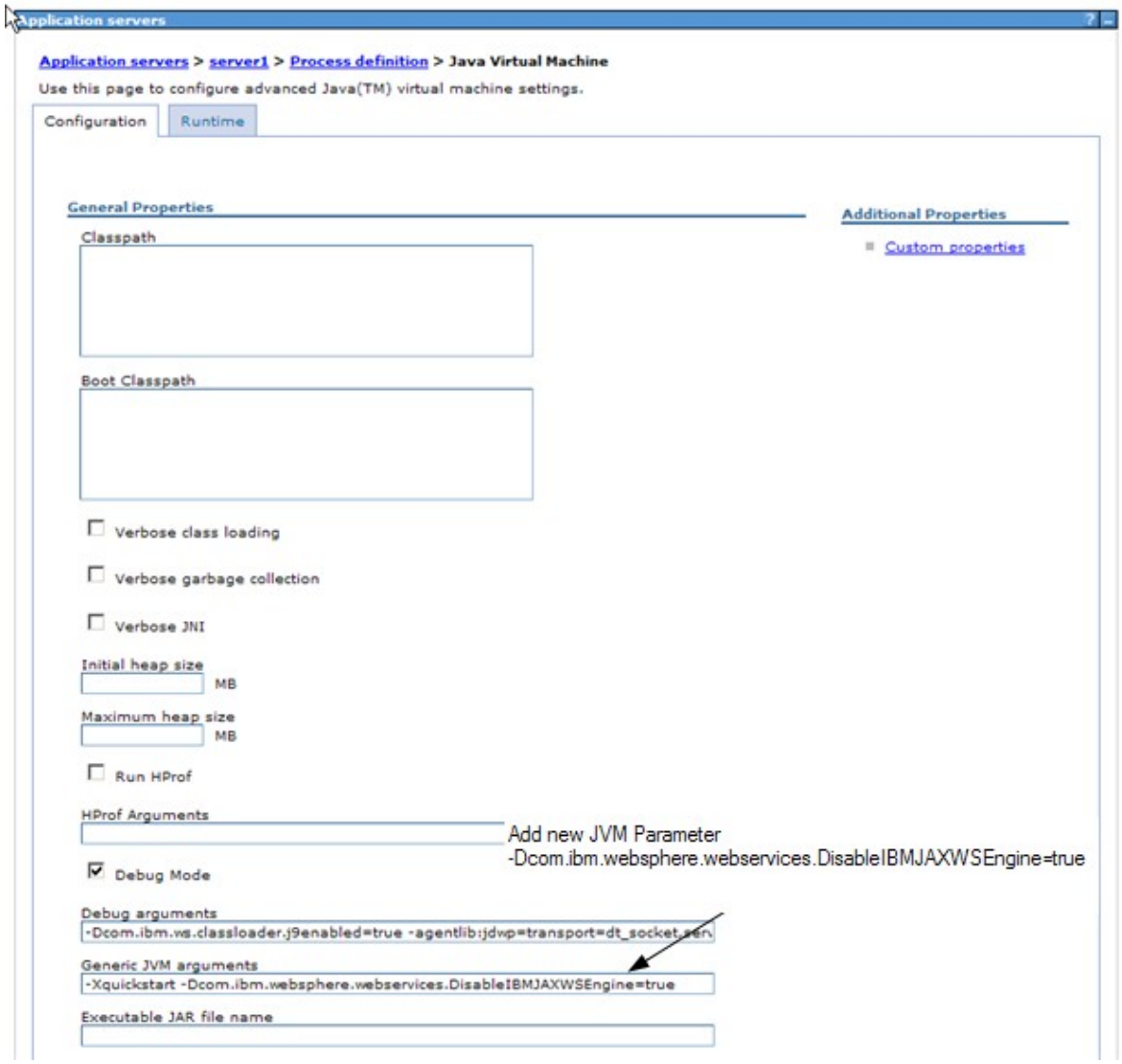
Disabling the IBM JAX-WS implementation

You need to disable the IBM JAX-WS implementation to ensure that Application Server doesn't scan for JAX-WS annotation and perform WebSphere-specific validations on the module that holds CXF application. You can turn off annotation scanning in one of two ways:

- To turn off annotation scanning for just the deployed module, modify the module's WebContent/META-INF/ MANIFEST.MF and add the following line `DisableIBMJAXWSEngine:true`.
- To turn off annotation scanning for the entire server, use the administrative console to navigate to the Java Virtual Machine, as shown below.



- As shown below, set the new JVM parameter to: `Dcom.ibm.websphere.webservices.DisableIBMJAXWSEngine=true`



Summary

In this article you learned how to deploy a Web services application built using third-party JAX-WS runtimes like Apache Axis2 and CXF on WebSphere Application Server V7.0. When building JAX-WS Web services applications on WebSphere, we recommend using the JAX-WS implementation provided by Websphere Application Server. However; if you choose to use third-party runtime, you now have an understanding of how to do that and the limitations of doing so.

Resources

- WebSphere Application Server V7 Information Center
- WebSphere Application Server V7: Understanding Class Loaders (PDF)
- JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0
- Axis-based applications fail to work properly in WebSphere Application Server V5.0.2 (Tech Note)
- Axis2 JAX-WS Engine
- Apache CXF
- developerWorks WebSphere Web services and SOA zone: Technical resources, such as articles, tutorials, and downloads for WebSphere Web services solutions.
- developerWorks WebSphere Application Server zone: Technical resources, such as articles, tutorials, and downloads, on WebSphere Application Server.

About the author

Nikhil Thaker is an Advisory Software Engineer with IBM Software Group, and a member of the WebSphere Application Server team that developed Web services JAX-WS implementation. He has more than nine years of experience in Enterprise Application Integration, and has focused on Web services for the past four years. He has worked with various IBM customers as an IT Specialist in Enterprise Application Integration in IBM Global Services. His industry exposure includes automotive, health care, telecommunication and utilities. You can reach Nikhil at nikhil.v.thaker@us.ibm.com.