

# JOnAS OpenSource Java EE Application Server - Doc - JAX-WS Integration

## [Preamble](#)

Preamble

Before going further, the reader should take a look at the Java EE web services specifications:

- JAX-WS 2.1 (JSR 224): <http://jcp.org/en/jsr/detail?id=224>
- WS MetaData 2.0 (JSR 181): <http://jcp.org/en/jsr/detail?id=181>
- WS for Java EE 1.2 (JSR 109): <http://jcp.org/en/jsr/detail?id=109>

Maybe a look at some implementation may worth:

- Axis2: <http://ws.apache.org/axis2>
- CXF: <http://cxf.apache.org>

Server side the JAX-WS server side part has to be able to expose POJO (from a web-app) and EJB (from an EjbJar) based web services to be exposed through HTTP (at least, that's what the spec requires).

## **Concepts**

WebService Endpoint A WS endpoint is a container around an endpoint implementation (Axis2 or CXF for example, but not limited to). It provides a management interface that an administrator can use to "explore" the web service. It also has a runtime interface that has to be used to invoke the web service. This runtime interface is a simple invoke method with 2 parameters (IWSRequest and IWSResponse) wrapping input and output messages streams. The IWSRequest and IWSResponse interfaces has to be implemented to adapt a given InputStream to the WS stream (like wrapping a HttpServletRequest and a HttpServletResponse for a webcontainer). WebService Deployment Manager A WS Deployment manager is an object that knows how to expose an endpoint using some protocol. For example, a deployment manager that we will probably have to implement is a web container based deployment manager: It will know what to do with the given endpoint to expose it through HTTP (by using a Servlet or anything else that can intercept the request/response flow and invoke the endpoint).

## **Changes**

Endpoint

I've added an EndpointType enumeration that sounds good for any WSEndpoint client to do something different if it is an EJB or POJO based endpoint. Along with this change, there is a getType():EndpointType method now in IWSEndpoint. Manager

I've done some refactoring on the DeploymentManager thing to unify POJO and EJB based deployment: we now only have register/unregister with an endpoint. Then, it's the manager's job to decide what to do, given the EndpointType of the IWSEndpoint. OSGi's whiteboard pattern (to be considered)

<http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf>

Using this pattern we could, when we discover a webservice, just register the IWSEndpoint instance as an OSGi service in the registry, so that, a DeploymentManager could use them. It means that we could provide WSEndpoint objects without first having a DeploymentManager, and when the manager becomes available, it will automatically deploy the existing web services. Client side

This part has not been discussed so far Flows

## **Endpoint Deployment Flow**

This part describes what happen when JOnAS have to deploy a web service. It's what have been discussed during the bootstrap face to face meeting held in Grenoble during May. IWSEndpoint creation

As a web service is not a deployable unit (in terms of Java E modules), it cannot have it's own IDeployer. A web service (endpoint, aka on the server side) can only be contained in an EjbJar or in a WebApp.

# JOnAS OpenSource Java EE Application Server - Doc - JAX-WS Integration

For EJB 3.x, EasyBeans provides some Callback mechanism that can be used to hook web services deployment with EJB3. It has the disadvantage to be EasyBeans's specific, but it provides us with almost all the informations we need (at least a reference to the EasyBeans's Factory).

With WebApp, there is no (currently) such Callback mechanism, so we have to provide one, or directly hook in the BaseWebContainerService code to analyze the web-app and creates the appropriate endpoints.

Once we are hooked inside on the general deployment process, endpoint creations can start. We have to analyze the classes metadata for some annotations, perform a merge if a webservicexml file is provided, then these informations are used to create and configure an IWebServiceEndpoint instance (**maybe the JAX-WS Service can be used as some kind of factory: getNewWebServiceEndpoint(?)**).

Once the endpoint has been created and configured we have 2 choices:

1. Use a given IWSDeploymentManager instance to register the endpoints
2. Register the endpoints in the OSGi ServiceRegistry (accordingly to the whiteboard pattern) so that any IWSDeploymentManager can use them.

IWSDeploymentManager's job

A web container based DeploymentManager (DM) has to expose any registered endpoints using a servlet for example. EJB

An EJB has no web interface, it can run without any web interface or web container, so it's the role of a web based DM to create a new Context (in Tomcat's terms) that will be configured to invoke a particular endpoint.

It has not be decided if we will have 1 context per endpoint or if multiple endpoints can share the same context. We still needs to define what will be the benefits of the differents approaches. POJO

As POJO based web services are declared inside a servlet, we already have the context, moreover, if the developer has provided a web.xml, maybe it has already defined some URL mapping that he wants to use for his endpoint (he can have configured web security too). So it's meaningless to create a new context where we already have one, and that is probably already configured from the developer's wishes.

So, a DM only has probably to configure the context:

- declaring servlets if needed
- specifying any required url-mappings
- set web security
- ...

All of this has to be done **BEFORE** the context is given to the web container. Tomcat/Jetty

As the programmatic deployment of new Context or webapps is specific for each web container, it seems OK that we will have 2 different DMs, one that knows how to configure Tomcat's internal for the endpoints, and the other that knows more about Jetty. Other DeploymentManager

As the DeploymentManager interface is protocol agnostic, it can be implemented using any protocol. For example, it should be easy to have the ability to provide a JMS based DM that will expose the endpoints using messaging technologies: ie when a JMS message hits a Destination, an Endpoint can be invoked asynchronously.

## ***Endpoint Request Flow***

**TO BE DESCRIBED** Remaining discussion items

## JOnAS OpenSource Java EE Application Server - Doc - JAX-WS Integration

We did not discussed about:

- Client side: service-ref
- WS MetaData analysis: this has probably to be done using the meta data facilities provided by the ow2-util project ( <http://fisheye.easybeans.org/browse/EasyBeans/trunk/util/modules/ee/metadata>)

[Preamble](#) (en)

Creator: xwiki:XWiki.sauthieg Date: 2008/05/19 09:16

Last Author: xwiki:XWiki.sauthieg Date: 2008/05/19 13:42

Copyright (c) 2006, [ObjectWeb Consortium](#)