# JBossWS Documentation

## 1. JBossWS Runtime and Tools support Overview

1.1. Key Features of JBossWS

1.2. Other relevant resources on the topic

## 2. Creating a Simple Web Service

2.1. Generation

## 3. Creating a Web Service using JBossWS runtime

3.1. Creating a Dynamic Web project

3.2. Configure JBoss Web Service facet settings

3.3. Creating a Web Service from a WSDL document using JBossWS runtime

3.4. Creating a Web service from a Java bean using JBossWS runtime

## 4. Creating a Web Service Client from a WSDL Document using JBoss WS

## 5. JBoss WS and development environment

5.1. JBossWS Preferences

5.2. Default Server and Runtime

## 6. Sample Web Service wizards

6.1. Sample Web Service

6.1.1. Generation

6.1.2. Deployment

6.2. Sample RESTful Web Service

## 7. RestEasy simple project example

7.1. The example project

## 8. Web Service Test View

8.1. Preliminaries

8.2. Testing a Web Service

8.3. Testing a RESTful Web Service

8.3.1. RestfulSample project

8.3.2. RESTEasy sample project
JBossWS Runtime and Tools support Overview

JBossWS is a web service framework developed as a part of the JBoss Application Server. It implements the JAX-WS and JAX-RS specifications. JAX-WS (Java API for XML Web Services) defines a programming model and run-time architecture for implementing web services in Java, targeted at the Java Platform, Enterprise Edition 5 (Java EE 5). JAX-RS (Java API for RESTful Web Services) is a Java API that supports the creation of Representational State Transfer (REST) web services, using annotations.

JBossWS integrates with most current JBoss Application Server releases as well as earlier ones, that did implement the J2EE 1.4 specifications. Even though JAX-RPC, the web service specification for J2EE 1.4, is still supported JBossWS does put a clear focus on JAX-WS.

JBossWS Tool works with JBossWS Runtime. Users can easily create, deploy and run a Web Service(WSDL base or RESTful base) and a Web Service Client using JBossWS Tool and JBossWS Runtime.

Also JBossWS Tool gives a way to test a web service running on a server.

1.1. Key Features of JBossWS

For a start, we propose you to look through the table of main features of JBossWS Runtime:

Table 1.1. Key Functionality for JBossWS

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAX-RPC and JAX-WS support</td>
<td>JBossWS implements both the JAX-WS and JAX-RPC specifications.</td>
</tr>
<tr>
<td>JAX-RS support</td>
<td>JBossWS implements the JAX-RS specification.</td>
</tr>
<tr>
<td>EJB 2.1, EJB3 and JSE endpoints</td>
<td>JBossWS supports EJB 2.1, EJB3 and JSE as Web Service Endpoints.</td>
</tr>
<tr>
<td>WS-Security 1.0 for XML Encryption/Signature of the SOAP message</td>
<td>WS-Security standardizes authorization, encryption, and digital signature processing of web services.</td>
</tr>
<tr>
<td>JBoss AS</td>
<td>JBoss Application Server 5 (JavaEE 5 compliant) web service stack.</td>
</tr>
<tr>
<td>Support for MTOM/XOP and SwA-Ref</td>
<td>Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) more efficiently serialize XML Infosets that have certain types of content.</td>
</tr>
</tbody>
</table>
1.2. Other relevant resources on the topic

You can find some extra information on:

- **JBossWS Wiki FAQ.** [http://www.jboss.org/community/wiki/JBossWS-FAQ#Tools]
Creating a Simple Web Service

This chapter describes how to create a simple web service.

2.1. Generation

A simple web service can be created by using the Simple Web Service wizard as described in Generate a simple web service

Procedure 2.1. Generate a simple web service

1. Access the New - Select a wizard dialog

   a. Right click on the project name in the Project Explorer view.

   b. Select New → Other.

   c. Expand the Web Services folder and click on the Simple Web Service option.

   Result: The New - Select a wizard dialog displays with the selected wizard type highlighted.
Figure 2.1. The New - Other (Wizard selection) dialog

2. **Access the Simple Web Service dialog**

   Click the **Next** button to proceed.

   **Result:** The Simple Web Service - Project and Web Service Details dialog displays.
3. **Define the service attributes**

Define the project, web service, package and class names according to the options displayed in [Table 2.1, “Project and Web Service Details”](#).

**Table 2.1. Project and Web Service Details**

<table>
<thead>
<tr>
<th>Dialog group</th>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>Technology</td>
<td>yes</td>
<td>Select the technology the Web Service will be based on.</td>
<td>A simple web service can be based on either the Web Service Definition Language (WSDL) or RESTful (REST) API. Click the radio button</td>
</tr>
</tbody>
</table>
Chapter 2. Creating a Simple ... 

<table>
<thead>
<tr>
<th>Dialog group</th>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic web project</td>
<td>yes</td>
<td>Select the project name.</td>
<td>The project name will default to the highlighted project in the Project Explorer. A different project can be selected from the drop-down list.</td>
<td></td>
</tr>
<tr>
<td>Service details</td>
<td>Service name</td>
<td>Enter the name to for the web service.</td>
<td>The web service name will be the URL for the service as mapped in the deployment descriptor (web.xml).</td>
<td></td>
</tr>
<tr>
<td>Update web.xml</td>
<td>no</td>
<td>Checkbox is checked by default, but is not mandatory.</td>
<td>Leaving this checked will add your new service to the web.xml in your project.</td>
<td></td>
</tr>
<tr>
<td>Service implementation</td>
<td>Package</td>
<td>yes</td>
<td>Enter the package for the web service servlet.</td>
<td>The default package is org.jboss.samples.webservices. Select your own package using the ... button.</td>
</tr>
<tr>
<td>Class</td>
<td>yes</td>
<td>Enter the name of the web service servlet.</td>
<td>The default class name will correspond to the default web service name resulting in an equivalent URL to servlet name mapping in the deployment descriptor (web.xml).</td>
<td></td>
</tr>
<tr>
<td>Application class</td>
<td>only</td>
<td>when the JAX-RS technology option is selected</td>
<td>The default application class is MyRESTApplication. Select your own application class using the ... button.</td>
<td></td>
</tr>
</tbody>
</table>

4. Generate the web service

Click the Finish button to complete the web service setup.

Result: The web service classes will be generated and the web.xml file updated with the deployment details if the Update web.xml option was selected.
Figure 2.3. Created Simple Web Service
Chapter 3.

Creating a Web Service using JBossWS runtime

In this chapter we provide you with the necessary steps to create a Web Service using JBossWS runtime. First you need to create a Dynamic Web project:

3.1. Creating a Dynamic Web project

Before creating a web service, you should have a Dynamic Web Project created:

![Figure 3.1. Dynamic Web Project](image)

Create a Web project by selecting **New > Project... > Dynamic Web project**. Enter the following information:

- **Project Name**: enter a project name
- **Target runtime**: any server depending on your installation. If it is not listed, click New button and browse to the location where it is installed to. You may set **Target Runtime** to **None**, in this case, you should read the section **Section 3.2, “Configure JBoss Web Service facet settings”**.
Figure 3.2. Dynamic Web Project Wizard

- Configuration: You may Section 3.2, “Configure JBoss Web Service facet settings” by clicking the Modify... button. The opened page is like Figure 3.4.

- Configure Web Module values:
3.2. Configure JBoss Web Service facet settings

If you have already created a new Dynamic Web project and not set the JBoss Web Service facet to the project, the next step is to add JBoss Web Service facet to the project. Right-click on the

Figure 3.3. Web Module Settings Configuration

If you added the JBoss Web Service facet to the project, now the Finish button is unavailable. You must click Next button to set more information about the JBoss Web Service facet. The page is like Figure 3.5. Then click on the Finish button.

If you didn't add the JBoss Web Service facet to the project, click on the Finish button. Next you will need to add JBoss Web Service facet to the project.
project, select its Properties and then find Project Facets in the tree-view on the left-side of the project properties dialog. Tick on the check box for JBoss Web Services. You will see what like this:

![Project Facets](image)

**Figure 3.4. Choose JBoss Web Service Facet**

At the bottom-left of the right-side of the project properties dialog, there is a error link: Further configuration required... . You must click the link to set more information about JBoss Web Service facet.

Click on the Further configuration required... link. In the opened window
Configure JBoss Web Service facet settings

Figure 3.5. Configure JBoss Web Service Facet

Server Supplied JBossWS Runtime: If you have already set a JBoss runtime to the project's target runtime, you may choose *Server Supplied JBossWS Runtime* and then click *Ok* to finish the configuration of JBoss Web Service facet.

If the project has no *Target Runtime* settings, you should check the second radio button and specify a JBossWS runtime from the list. You also can create a new JBossWS runtime, click on the *New...* button will bring you to another dialog to configure new JBossWS runtime.
3.3. Creating a Web Service from a WSDL document using JBossWS runtime

In this chapter we provide you with the necessary steps to create a Web Service from a WSDL document using JBossWS runtime.

Make sure that you have already created a dynamic Web project with JBoss Web Service facet installed and set the necessary preferences through the Preference menu.

See how to make it in the Section 3.1, “Creating a Dynamic Web project”, Section 3.2, “Configure JBoss Web Service facet settings” and Chapter 5, JBoss WS and development environment sections.

To create a Web Service using JBossWS runtime select File > New > Other > Web Services > Web Service to run Web Service creation wizard.

Let's get through the wizard step-by-step:
Figure 3.7. New Web Service Wizard

• Select the stages of Web service development that you want to complete using the slider:
  
  • Develop: this will develop the WSDL definition and implementation of the Web service. This includes such tasks as creating modules that will contain generated code, WSDL files, deployment descriptors, and Java files when appropriate.
• Assemble: this ensures the project that will host the Web service or client gets associated to an EAR when required by the target application server.

• Deploy: this will create the deployment code for the service.

• Install: this will install and configure the Web module and EARs on the target server.

• Start: this will start the server once the service has been installed on it. The server-config.wsdd file will be generated.

• Test: this will provide various options for testing the service, such as using the Web Service Explorer or sample JSPs.

• Select your server: the default server is displayed. If you want to deploy your service to a different server click the link to specify a different server.

• Select your runtime: ensure the JBoss WS runtime is selected.

• Select the service project: the project selected in your workspace is displayed. To select a different project click on the project link. If you are deploying to JBoss Application Server you will also be asked to select the EAR associated with the project. Ensure that the project selected as the Client Web Project is different from the Service Web Project, or the service will be overwritten by the client’s generated artifacts.

• If you want to create a client, select the type of proxy to be generated and repeat the above steps for the client. The better way is to create a web service client project separately.

First, select Top down Java bean Web Service from the Web Service type list, and select a WSDL document from workspace, click on the Server name link on the page will bring you to another dialog. Here you can specify the server to a JBoss Server and Web Service runtime to JBossWS runtime:
Figure 3.8. Select Server and Web Service runtime

Click on the *Finish* button to see the next wizard view opened:
Figure 3.9. New Web Service Wizard

Click on the Next button to proceed:
Figure 3.10. New Web Service Wizard

On this page, the default package name comes from the namespace of the WSDL document, you also can change it to any valid package name you want. JAX-WS specification should be set to 2.0 if your JBossWS runtime in JBoss Server is JBossWS native runtime. You can specify a catalog file and binding files if you have them. If you want the wizard to generate empty implementation
classes for the Web Service, check the **Generate default Web Service implementation classes** check box. If you want to update the default Web.xml file with the Web Service servlets configured, check the **Update the default Web.xml** check box. Click on the **Next** or on the **Finish** button to generate code.

Once the Web Service code is generated, you can view the implementation class and add business logic to each method.

---

**Figure 3.11. The generated implementation Java code**

View the Web.xml file:
In the next chapter you will find out how to create a Web service from a Java bean.

3.4. Creating a Web service from a Java bean using JBossWS runtime

To create a Web service from a bean using JBoss WS:

Setup Chapter 5, JBoss WS and development environment.

Create Section 3.1, “Creating a Dynamic Web project”.
Chapter 3. Creating a Web Ser...

Note

To use the Simple Web Service wizard to create this Web Service, replace the Class and Application Class fields with your specific classes, within the instructions in Chapter 2, Creating a Simple Web Service.

Section 3.2, “Configure JBoss Web Service facet settings”

Create a Web Service from a java bean:

- Switch to the Java EE perspective Window → Open Perspective → Java EE.
- In the Project Explorer view, select the bean that you created or imported into the source folder of your Web project.
Figure 3.13. Select the Bean Created

- Click **File → New → Other**. Select Web Services in order to display various Web service wizards. Select the Web Service wizard. Click on the Next button.
Figure 3.14. New Web Service

- On the first Web Service wizard page: select Bottom up Java bean Web service as your Web service type, and select the Java bean from which the service will be created:
Creating a Web service from a Java bean using JBossWS runtime

Figure 3.15. Set Web Service Common values

Click on the Next button.

- On the JBoss Web Service Code Generation Configuration page, set the following values:
Figure 3.16. Set Web Service values for Code Generation

- Generate WSDL file: select it, you will get a generated WSDL file in your project. But this wsdl’s service address location values are not a real address.

- After the Web service has been created, the following option can become available depending on the options you selected: Update the default web.xml file. If selected, you may test the web service by Explorer.

Click on the Next button.

- On this page, the project is deployed to the server. You can start the server and test the web service. If you want to publish the web service to a UDDI registry, you may click the Next button to publish it. If not, you may click the Finish button.
Figure 3.17. Start a Server

After the Web Service has been created, the following options may become available depending on the options selected:

- the generated web services code
- If you selected to generate a WSDL file, you will get the file in your project's wsdl folder.
Figure 3.18. The Generated HelloWorldService.wsdl File in the wsdl Folder

- If you selected to update the default web.xml, you will test the web service in the browser. Open the Explorer, input the url for the web service according to web.xml plus ?wsdl, you will get the WSDL file from Explorer.
Figure 3.19. The Updated web.xml file

In the next chapter you will be able to create a Web Service Client from a WSDL document using JBoss WS.
Creating a Web Service Client from a WSDL Document using JBoss WS

To create a Web Service Client from a WSDL Document using JBoss WS you need to fulfill the following steps:

Setup Chapter 5, JBoss WS and development environment.

Section 3.1, “Creating a Dynamic Web project”.

Section 3.2, “Configure JBoss Web Service facet settings”.

Then you can create a Web Service Client from a WSDL document:

• Switch to the Java EE perspective Window > Open Perspective > Java EE.

• Click File > New > Other. Select Web Services in order to display the various Web service wizards. Select the Web Service Client wizard. Click on the Next button.

![Figure 4.1. New Web Service Client](image-url)
The first and the second Web Service Client wizard pages are the same as for Section 3.3, “Creating a Web Service from a WSDL document using JBossWS runtime”.

Figure 4.2. Set Web Service Common values
**Figure 4.3. Set Web Service values related to WSDL file**

The only difference is:

- **Client Type:** Support of Java Proxy only.

Click on the Finish button.

After the Web Service Client has been created, the following may occur depending on the options you selected:

- the generated web service and client codes
- a client sample class.
Figure 4.4. Client Sample Class

JBoss WS use a Java class to test Web Service. A client sample class will be generated, you may run this client as a java application to call a web service.

**Note:**

To run client sample as a Java application you need a JBoss Runtime in build path.
Chapter 5.

JBoss WS and development environment

In this chapter you will learn how to change JBossWS preferences and how to set default server and runtime.

5.1. JBossWS Preferences

In this section you will know how JBossWS preferences can be modified during the development process.

JBossWS preferences can be set on the JBossWS preference page. Click on Window > Preferences > JBoss Tools > Web > JBossWS Preferences.

On this page you can manage the JBossWS Runtime. Use the appropriate buttons to Add more runtimes or to Remove those that are not needed.
Figure 5.1. JBossWS Preferences Page

Clicking on *Add* or *Edit* button will open the form where you can configure a new JBossWS runtime and change the path to JBossWS runtime home folder, modify the name and version of the existing JBossWS runtime settings. Press Finish to apply the changes.
Figure 5.2. Edit JBossWS Runtime

WS container allows Source and JavaDoc locations to be set via the Properties dialog on each contained .jar: right-click on any .jar file in the Project Explorer view, select Properties. Choose Java Source Attachment and select location (folder, JAR or zip) containing new source for the chosen .jar using one of the suggested options (workspace, external folder or file) or enter the path manually:
Figure 5.3. Classpath Container: Java Source Attachment

Click on Apply and then on Ok.

To change JavaDoc Location choose Javadoc Location and specify URL to the documentation generated by Javadoc. The Javadoc location will contain a file called package-list:
Figure 5.4. Classpath Container: Javadoc Location

Click on Apply and then on Ok.

5.2. Default Server and Runtime

Open Window → Preferences → Web Services → Server and Runtime. On this page, you can specify a default server and runtime.

For ease of use, the better way is to set runtime to JBoss WS.

After server and runtime are specified, click on the Apply button to save the values.
Figure 5.5. Specifying a default server and runtime
Sample Web Service wizards

JBoss Tools includes wizards for the creation of sample web services. These include:

- **Create a sample Web Service** for a JAX-WS web service; and
- **Create a sample RESTful Web Service** for a JAX-RS web service.

These wizards are used within a Dynamic Web project. A dynamic web project can be created by following the steps in *Creating a dynamic web project*.

**Procedure 6.1. Creating a dynamic web project**

1. **Access the New Project Dialog**
   
   Select **File → New → Project**
   
   **Result:** The New Project screen displays.

2. **Define the Project Type**
   
   a. Click the Dynamic Web Project label by expanding the Web folder.
   
   b. Click the Next button to proceed.
   
   **Result:** The New Dynamic Web Project screen displays.
Define the Project Attributes

Define the Dynamic Web Project attributes according to the options displayed in Table 6.1, “New Dynamic Web Project”
### Table 6.1. New Dynamic Web Project

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project name</td>
<td>yes</td>
<td>Enter the project name.</td>
<td>The project name can be any name defined by the user.</td>
</tr>
<tr>
<td>Project location</td>
<td>yes</td>
<td>Click the <strong>Use default location</strong> checkbox to define the project location as the Eclipse workspace or define a custom path in the <strong>Location</strong> field.</td>
<td>The default location corresponds to the Eclipse workspace.</td>
</tr>
<tr>
<td>Target runtime</td>
<td>no</td>
<td>Select a pre-configured runtime from the available options or configure a new runtime environment.</td>
<td>The target runtime defines the server to which the application will be deployed.</td>
</tr>
<tr>
<td>Dynamic web module version</td>
<td>yes</td>
<td>Select the required web module version.</td>
<td>This option adds support for the Java Servlet API with module versions corresponding to J2EE levels as listed in Table 6.2, <em>New Dynamic Project - Dynamic web module version</em>.</td>
</tr>
<tr>
<td>Configuration</td>
<td>yes</td>
<td>Select the project configuration from the available options.</td>
<td>The project can be based on either a custom or a set of pre-defined configurations as described in Table 6.3, <em>New Dynamic Project - Configuration</em>.</td>
</tr>
<tr>
<td>EAR membership</td>
<td>no</td>
<td>Add the project to an existing EAR project.</td>
<td>The project can be added to an existing EAR project by selecting the checkbox. Once checked, a new EAR project can be defined by clicking the <strong>New Project</strong> button.</td>
</tr>
<tr>
<td>Working sets</td>
<td>no</td>
<td>Add the project to an existing working set.</td>
<td>A working set provides the ability to group projects or project attributes in a customized way to improve access. A new working set can be defined once the <strong>Select</strong> button has been clicked.</td>
</tr>
</tbody>
</table>
Table 6.2. New Dynamic Project - Dynamic web module version

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>This web module version corresponds to the J2EE 1.2 implementation.</td>
</tr>
<tr>
<td>2.3</td>
<td>This web module version corresponds to the J2EE 1.3 implementation.</td>
</tr>
<tr>
<td>2.4</td>
<td>This web module version corresponds to the J2EE 1.4 implementation.</td>
</tr>
<tr>
<td>2.5</td>
<td>This web module version corresponds to the JEE 5 implementation.</td>
</tr>
</tbody>
</table>

Table 6.3. New Dynamic Project - Configuration

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;custom&gt;</td>
<td>Choosing from one of the pre-defined configurations will minimise the effort required to set up the project.</td>
</tr>
<tr>
<td>BIRT Charting Web Project</td>
<td>A project with the BIRT Charting Runtime Component.</td>
</tr>
<tr>
<td>BIRT Charting Web Project</td>
<td>A project with the BIRT Reporting Runtime Component.</td>
</tr>
<tr>
<td>CXF Web Services Project v2.5</td>
<td>Configures a project with CXF using Web Module v2.5 and Java v5.0.</td>
</tr>
<tr>
<td>Default Configuration for JBoss 5.0 Runtime</td>
<td>This option is a suitable starting point. Additional facets can be installed later to add new functionality.</td>
</tr>
<tr>
<td>Dynamic Web Project with Seam 1.2</td>
<td>Configures a project to use Seam v1.2.</td>
</tr>
<tr>
<td>Dynamic Web Project with Seam 2.0</td>
<td>Configures a project to use Seam v2.0.</td>
</tr>
<tr>
<td>Dynamic Web Project with Seam 2.1</td>
<td>Configures a project to use Seam v2.1.</td>
</tr>
<tr>
<td>Dynamic Web Project with Seam 2.2</td>
<td>Configures a project to use Seam v2.2.</td>
</tr>
<tr>
<td>JBoss WS Web Service Project v3.0</td>
<td>Configures a project with JBossWS using Web Module v2.5 and Java v5.0.</td>
</tr>
<tr>
<td>JavaServer Faces v1.2 Project</td>
<td>Configures a project to use JSF v1.2.</td>
</tr>
<tr>
<td>Minimal Configuration</td>
<td>The minimum required facets are installed. Additional facets can be chosen later to add functionality to the project.</td>
</tr>
</tbody>
</table>

4. **Access the Java sub-dialog**

Click **Next** to proceed.

**Result:** The **New Dynamic Web Project - Java** dialog displays.
5. **Define the source and output folders**

Define the Dynamic Web Project source and output folders by adding or editing folders as required.

6. **Access the Web Module sub-dialog**

Click **Next** to proceed.

**Result:** The **New Dynamic Web Project - Web Module** dialog displays.

![Image of New Dynamic Web Project - Web Module dialog]

**Figure 6.2. New Dynamic Web Project - Web Module**

7. **Enter the web module settings**

Define the settings as listed in **Table 6.4, “New Dynamic Web Project - Web Module”** including the root folder for path names in the web project context and the name of the web content directory.

**Table 6.4. New Dynamic Web Project - Web Module**

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context root</td>
<td>yes</td>
<td>Enter the context root for the project.</td>
<td>The context root identifies a web application to the server and which URLs to delegate to the application.</td>
</tr>
<tr>
<td>Content directory</td>
<td>yes</td>
<td>Enter the directory name for the web content.</td>
<td>Web resources such as html, jsp files and graphic files will be written to the specified content directory.</td>
</tr>
</tbody>
</table>
Chapter 6. Sample Web Service...

### 6.1. Sample Web Service

These sections describe how to generate and deploy a sample web service.

#### 6.1.1. Generation

A sample web service can be created by using the Create a Sample Web Service wizard as described in *Generate a sample web service*

**Procedure 6.2. Generate a sample web service**

1. **Access the New - Select a wizard dialog**

   a. Right click on the project name in the Project Explorer view.

   b. Select New → Other.

   c. Click the Create a Sample Web Service label by expanding the Web Services folder.

   **Result:** The New - Select a wizard dialog displays with the selected wizard type highlighted.

2. **Access the Generate a Sample Web Service dialog**

   Click the Next button to proceed.

   **Result:** The Generate a Sample Web Service - Project and Web Service Name dialog displays.

---

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate web.xml deployment</td>
<td>no</td>
<td>Check this box to generate a deployment descriptor for the project.</td>
<td></td>
</tr>
<tr>
<td>descriptor</td>
<td></td>
<td></td>
<td>URL to servlet mappings and servlet authentication details are written to the deployment descriptor enabling the web server to serve requests.</td>
</tr>
</tbody>
</table>

---

8. **Open the Java EE perspective.**

   a. Click the Finish button to complete the project setup.

   **Result:** If not already set, a dialog will appear prompting the user to open the relevant perspective.

   b. Click the Yes button to display the Java EE perspective.

   **Result:** The project is configured and the Java EE perspective is displayed.
3. **Define the service attributes**

Define the project, web service, package and class names according to the options displayed in Table 6.5, "Project and Web Service Name"

**Table 6.5. Project and Web Service Name**

<table>
<thead>
<tr>
<th>Dialog group</th>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Web Project</td>
<td>Name</td>
<td>yes</td>
<td>Enter the project name.</td>
<td>The project name will default to the highlighted project in the Project Explorer. A different project can be selected from the list or entered directly in the editable drop-down list.</td>
</tr>
<tr>
<td>Web Service</td>
<td>Name</td>
<td>yes</td>
<td>Enter the name for the web service.</td>
<td>The web service name will be the url for the service as mapped in the deployment descriptor (web.xml).</td>
</tr>
</tbody>
</table>
4. **Generate the web service**

Click the **Finish** button to complete the web service setup.

**Result:** The web service classes will be generated and the web.xml file updated with the deployment details.

5. **Browse the HelloWorld.java class**

Double click the HelloWorld.java class and note the annotated class name and method. These annotations identify the web service entities to the server.

![HelloWorld.java](image)

**Figure 6.4. web.xml**

6. **Browse the web.xml deployment descriptor**

Double click the web.xml file and note the servlet mapping as defined in **Figure 6.3, “Generate a Sample Web Service - Project and Web Service Name”**. Note also that:
• the main servlet for the application is `org.jboss.samples.webservices.HelloWorld`
  which is given the custom name `HelloWorld`; and

• the main servlet is mapped to the particular url `/HelloWorld` [1].

Figure 6.5. `web.xml`

Upon start up, the server will write a WSDL file to the `server-profile/data/wsdl/` directory

6.1.2. Deployment

Once created, the sample web service can be deployed to the target runtime as described in

*Export the project as a Web Archive (WAR)*.

Procedure 6.3. Export the project as a Web Archive (WAR)

1. **Access the Export dialog**

   a. Right click on the project name in the **Project Explorer** view.

   b. Select **Export → WAR file**.

   **Result:** The **Export- WAR Export** dialog displays with the selected web project
      highlighted.
Chapter 6. Sample Web Service...

Figure 6.6. Export - WAR Export dialog

2. **Complete the export dialog**

Define the WAR file attributes as described in *Table 6.6, “Export - War Export”*

**Table 6.6. Export - War Export**

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web project</td>
<td>yes</td>
<td>Enter the web project name.</td>
<td>The project name will default to the highlighted project in the <strong>Project Explorer</strong>. A different project can be selected from the list or entered directly in the editable drop-down list.</td>
</tr>
<tr>
<td>Destination</td>
<td>yes</td>
<td>Enter or browse to the destination.</td>
<td>Set the destination as the <strong>build</strong> folder to store the WAR file within the project. Alternatively, deploy the project directly to the <strong>deploy</strong> folder.</td>
</tr>
</tbody>
</table>
### Field | Mandatory Instruction | Description
--- | --- | ---
Optimize for a specific server runtime | no | Select this box to optimize the WAR file for deployment to the targeted runtime.

| directory of the target server profile.

The list of available runtimes will be those configured during the project set-up or by selecting File → New → Server.

3. **Deploy the application**

Copy the file to the `deploy` directory of the required target server profile, such as the all profile. Note that the WAR file destination may have already been set as the deploy directory in *Step 2*.

### 6.2. Sample RESTful Web Service

A sample Restful web service can be generated by following the steps outlined in *Generate a sample RESTful web service*.

**Procedure 6.4. Generate a sample RESTful web service**

**Target runtime must have RESTEasy installed**

The sample RESTful web service will not work unless it is deployed to a server with RESTEasy installed.

1. **Access the New - Select a wizard dialog**

   a. Right click on the project name in the **Project Explorer** view.
   
   b. Select **New → Other**.
   
   c. Click the **Create a Sample RESTful Web Service** label by expanding the **Web Services** folder.

   **Result:** The **New - Select a wizard** dialog displays with the selected wizard type highlighted.

2. **Access the Generate a Sample RESTful Web Service dialog**

   Click the **Next** button to proceed.

   **Result:** The **Generate a Sample RESTful Web Service - Project and Web Service Name** dialog displays.
3. **Define the service attributes**

Define the project, web service, package and class names according to the options displayed in *Table 6.7, “Project and Web Service Name”*

<table>
<thead>
<tr>
<th>Dialog group</th>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Web Project</td>
<td>yes</td>
<td>Enter the project name.</td>
<td>The project name will default to the highlighted project in the <strong>Project Explorer</strong>. A different project can be selected from the list or entered directly in the editable drop-down list.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Service</td>
<td>Name yes</td>
<td>Enter the name for the web service.</td>
<td>The web service name will be the url for the service as mapped in the deployment descriptor (<em>web.xml</em>).</td>
<td></td>
</tr>
<tr>
<td>Dialog group</td>
<td>Field</td>
<td>Mandatory</td>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Update web.xml</td>
<td>Field</td>
<td>No</td>
<td>Check this box to add the service to the deployment descriptor.</td>
<td>This option is checked by default and may be unchecked when deploying to JBoss AS 6.0 or RESTEasy 2.0 servers. Service information is not required in the deployment descriptor for these servers.</td>
</tr>
<tr>
<td>Sample Web Service Class</td>
<td>Package</td>
<td>Yes</td>
<td>Enter the package for the web service class.</td>
<td>The default package for the sample web service will be displayed.</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>Yes</td>
<td>Enter the name of the web service class containing the JAX-RS annotated path.</td>
<td>This class defines the path to the web service and is referenced in the Application Class Name. The Application Class Name is declared in the deployment descriptor providing indirect access to the annotated path.</td>
</tr>
<tr>
<td></td>
<td>Application Class Name</td>
<td>Yes</td>
<td>Enter the name of the Application Class Name.</td>
<td>The Application Class Name constructor instantiates objects of the web service class containing the JAX-RS annotated path, GET and POST methods. It serves as a single point of access to the application for the web server.</td>
</tr>
</tbody>
</table>

4. **Generate the web service**

   Click the **Finish** button to complete the web service setup.

   **Result:** The web service classes will be generated and the web.xml file updated with the deployment details.

5. **Browse the MyRESTApplication.java class**

   Double click the **MyRESTApplication.java** class and note the constructor instantiating objects of type HelloWorldResource. The relevance of this will be discussed shortly.
Figure 6.8. Application Class - MyRESTApplication.java

6. **Browse the HelloWorldResource.java class**

   Double click the `HelloWorldResource.java` class and note the JAX-RS annotated path and the annotated GET method.

Figure 6.9. HelloWorldResource.java

7. **Browse the web.xml deployment descriptor**

   Double click the `web.xml` file and note the `jax.ws.rs.Application` parameter mapped to the `Application` class. Note also that:
Sample RESTful Web Service

- the main servlet for the application is org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher which is given the custom name Resteasy; and

- the main servlet is not mapped to a particular url as indicated by /*.

The url for sending GET requests can be resolved as follows:

a. Identify the Application Class as defined in the deployment descriptor.

b. Note the object type instantiated in the Application class and added to the singleton set: HelloWorldResource.

c. Note the JAX-RS annotated path declared in the corresponding HelloWorldResource class: @Path("/MyRESTApplication") [1].

Figure 6.10. web.xml

The url for sending GET requests is therefore http://localhost:8080/ProjectName/[1] or, http://localhost:8080/RestfulSample/MyRESTApplication.
RestEasy simple project example

JBoss Tools includes many example projects which are available by selecting Help → Project Examples. The following sections describe setting up the example RESTEasy project. This project serves as a good example for testing the numerous Web Service Test View functions.

7.1. The example project

Once the required plugins have been installed, the example project can be set up as described in JBoss Tools New Example Project

Procedure 7.1. JBoss Tools New Example Project

1. Access the New Example Project Dialog
   Select Help → Project Examples
   Result: The New Example Project dialog displays.

2. Define the Example Project Type
   a. Click the RESTEasy Simple Example label by expanding the RESTEasy node.
   b. Click the Finish button to complete the project set up.
   Result: The simple project is configured and ready to build.

Project requirements
In the event that a message is displayed indicating some requirements could not be configured, click the Details button followed by the Fix button to rectify the problem. The message will be displayed as a result of missing plugins or a requirement to select or configure a suitable runtime.

3. Build the project
   Right click on the project name and select Run As → Maven package
   Result: The simple.war file is written to the project's target directory.

4. Deploy the project
   Copy the simple.war file to the deploy directory of the required server profile such as the all profile.
5. **Determine the URL for the web service**

Double click the `web.xml` file and note the `jax.ws.rs.Application` parameter mapped to the **Application** class. Note also that:

- the main servlet for the application is `org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher` which is given the custom name Resteasy; and
- the main servlet is mapped to the url `/rest-services/*` [1].

The url for sending GET requests can be resolved as follows:

a. Identify the **Application** class as defined in the deployment descriptor.

b. Note the object type (**CustomerResource**) instantiated in the **Application** class (**ShoppingApplication**) and added to the singleton set (**singletons.add(new CustomerResource())**).

c. Note the JAX-RS annotated path declared in the corresponding **CustomerResource class**: @Path("/customers") [2].

![Figure 7.1. web.xml](image)

The url for sending GET requests can be formed from `http://localhost:8080/ProjectName/[1]/[2]` or, `http://localhost:8080/simple/rest-services/customers..`
Web Service Test View

JBoss Tools provides a view to test web services. The Web Services Test View can be displayed by following the steps in Web Services Test View.

Procedure 8.1. Web Services Test View

- Access the Show View dialog
  
  a. Select Window → Show View → Other

  Result: The Show View dialog displays.

  b. Click on the Web Services Tester label by expanding the JBoss Tools Web Services node and click OK.

  Result: The Web Services test view displays.

![Figure 8.1. Web Service Test View](image)

The main components of the Web Service Tester View are:

- WSDL path/button bar (Table 8.1, “WSDL path/button bar”)
- Request details panel (Table 8.2, “Request details panel”)
- Response details panel (Table 8.3, “Response details panel”)
### Table 8.1. WSDL path/button bar

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editable dropdown list</td>
<td>Enter the location of the WSDL file or HTTP address of the service to be tested. The combo box requires the path to the WSDL in a URI format.</td>
</tr>
<tr>
<td>Combo box</td>
<td>Select the type of service to test. The options are <strong>JAX-WS</strong> or any other option to test a <strong>JAX-RS</strong> service using HTTP request methods (<strong>PUT, GET, POST, DELETE or OPTIONS</strong>).</td>
</tr>
<tr>
<td>Toolbar button - Get From WSDL</td>
<td>Click this button to display the <strong>Select WSDL</strong> dialog. Enter the <strong>URL, File system</strong> location or Eclipse <strong>Workspace</strong> location of the WSDL file. Given a valid file, the dialog will allow selection of the <strong>Port</strong> and <strong>Operation</strong> to test. Once selected, the request details will be displayed in the <strong>Request Details</strong> panel.</td>
</tr>
<tr>
<td>Toolbar button - Invoke</td>
<td>Once the WSDL file has been selected, the service can be invoked by clicking this button. Reponse details will be displayed in the <strong>Response Details</strong> panel.</td>
</tr>
</tbody>
</table>

### Table 8.2. Request details panel

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt for Basic Authentication</td>
<td>Select this check box to send a username and password with the request. Entering the user details for each subsequent request is not necessary as the details are stored in memory.</td>
</tr>
<tr>
<td>Headers</td>
<td>Enter (<strong>Add</strong>) one or more <strong>name=value</strong> pairs. These headers will be passed with the invocation request at the HTTP level where possible.</td>
</tr>
<tr>
<td>Parameters</td>
<td>As for header information, enter one or more <strong>name=value</strong> pairs by clicking the <strong>Add</strong> button.</td>
</tr>
<tr>
<td>Body</td>
<td>Enter the JAX-WS SOAP request messages or input for JAX-RS service invocations in this text box.</td>
</tr>
</tbody>
</table>

### Table 8.3. Response details panel

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td>The headers returned by the service invocation will be displayed in this panel.</td>
</tr>
<tr>
<td>Response body</td>
<td>The JAX-WS and JAX-RS response bodies will be displayed in this box. The raw text returned form the web service invocation can be displayed by clicking the <strong>Show Raw</strong> button. The output will be embedded in a html browser by clicking the <strong>Show in Browser</strong> button.</td>
</tr>
</tbody>
</table>
The following sections describe testing JAX-WS and JAX-RS web services including the necessary preliminary steps.

8.1. Preliminaries

The following procedure describes the steps to perform before testing a web service.

Procedure 8.2. Testing a JAX-RS web service

- **Preliminary steps**

Prior to testing a web service:

a. The **Web Service Test View** should be opened as described in **Web Services Test View**;

Result: The **Web Service Test View** displays.

![Figure 8.2. Web Service Test View](image)

b. A web service has been deployed to the **deploy** directory of the chosen server profile as described in:

- Export the project as a Web Archive (WAR) for the **RestfulSample** project; or
• JBoss Tools New Example Project for the RESTEasy sample project.

c. The server has been started with run.sh -c <profile>

8.2. Testing a Web Service

A JAX-WS web service can be tested by using the Web Service Tester View displayed in Figure 8.1, "Web Service Test View". The JAX-WS test is specified by:

1. Selecting the JAX-WS combobox option.

2. Entering the location of the WDSL file.

Step 2 can be performed in a number of ways including:

• entering the location directly in the editable dropdown list; or

• clicking the Get from WSDL file button and entering the URL, Eclipse workspace or File system details.

Testing a JAX-WS web service demonstrates testing the WebServiceSample project developed in Generate a sample web service.

Procedure 8.3. Testing a JAX-WS web service

1. Following the preliminary steps described in Testing a JAX-RS web service, select JAX-WS from the available combo box options.

   Result: The SOAP message details are displayed in the Request Body textbox of the Request Details panel.

   ▼ Request Body
   
   ```xml
     <soap:Header>
     </soap:Header>
     <soap:Body>
       </soap:Body>
     </soap:Envelope>
   ```

Figure 8.3. JAX-WS Body Text
2. Enter the location of the WSDL file in the editable dropdown list. The location for the WebServiceSample web service is [http://localhost:8080/WebServiceSample/HelloWorld?WSDL](http://localhost:8080/WebServiceSample/HelloWorld?WSDL)

3. Click the **Invoke** button.

**Result:** The Select WDSL dialog appears.

![Select WSDL dialog](image)

**Figure 8.4. Select WSDL**

4. **Select the required service attributes**

   Select the **Service**, **Port** and **Operation** from the comboboxes and click **OK**.

**Results:** The `<soap:Body/>` section of the SOAP message is filled with the SayHello message details.

```xml
<soap:Body>
<webs:sayHello xmlns:webs="http://webservices.samples.jboss.org/">
  <!-- optional -->
  <arg0>?</arg0>
</webs:sayHello>
</soap:Body>
```

**Figure 8.5. JBoss Tools Project Creation**

The response header details are returned.
8.3. Testing a RESTful Web Service

Testing a RESTful (JAX-RS) web service is achieved by following a similar procedure to testing a JAX-WS web service. Instead of selecting the JAX-WS option in the combobox, the JAX-RS service is invoked by sending HTTP method requests of the form OPTIONS, GET, POST, PUT and DELETE. As there is no WSDL file associated with a JAX-RS service, the available options can be determined by selecting OPTIONS in the combobox.

A JAX-RS web service can be tested by using the Web Service Tester View displayed in Figure 8.1, “Web Service Test View”. The JAX-RS test is specified by:

1. Selecting the OPTIONS combobox option.

2. Entering the url of the JAX-RS web service.
The test procedure is discussed in the following sections for both the RestfulSample and the RESTEasy sample projects developed earlier.

### 8.3.1. RestfulSample project

**Procedure 8.4. RestfulSample test**

1. **a. Query the available options**
   
   Select OPTIONS from the available combobox options.

   **b.** Enter the url of the web service in the editable drop-down list: http://localhost:8080/RestfulSample/MyRESTApplication.

   **c.** Click the Invoke button

   **Result:** The Response Headers text area indicates that the allowed options are [GET, OPTIONS, HEAD] as shown in Figure 8.8, “JAX-RS Response Header Text”.

   ![Response Details]

   **Response Headers**

   - [HTTP/1.1 200 OK]
   - Date=[Mon, 25 Oct 2010 11:12:24 GMT]
   - Content-Length=[0]
   - Allow=[GET, OPTIONS, HEAD]

   **Figure 8.8. JAX-RS Response Header Text**

2. **Test the GET request**

   **a.** Having established that the GET request is valid, select GET from the available combobox options.

   **b.** Click the Invoke button.

   **Result:** The Response Body text area displays the expected “Hello World” text as shown in Figure 8.9, “JAX-RS Response Body Text”.

   ![Response Details]

   **Response Headers**

   - [HTTP/1.1 200 OK]
   - Date=[Mon, 25 Oct 2010 11:12:24 GMT]
   - Content-Length=[0]
   - Allow=[GET, OPTIONS, HEAD]

   **Figure 8.9. JAX-RS Response Body Text**
Chapter 8. Web Service Test View

8.3.2. RESTEasy sample project

Procedure 8.5. Testing a JAX-RS web service- POST and GET requests

1. a. Query the available options

Following the preliminary steps described in Testing a JAX-RS web service, select the OPTIONS method from the operations text area.

b. Enter the url of the web service in the editable drop-down list http://localhost:8080/simple/rest-services/customers.

c. Click the Invoke button

Result: The Response Headers text area indicates that the allowed options are [POST, OPTIONS] as shown in Figure 8.10, “JAX-RS RESTEasy project Body Text”.

2. Test the POST option

a. Select POST method in the the operations drop-down list.
b. We will post xml data to this particular web service. Complete the header details by entering `content-type=application/xml` in the text area and click Add to add it to the Headers list.

**Result:** The **content-type** is added to the Headers list as shown in Figure 8.11, “content-type header”.

![Headers](image)

**Figure 8.11. content-type header**

**c. Enter customer details**

Enter the customer details in the Body Text area as displayed in Figure 8.12, “Customer data”.

```
<customer>
  <first-name>Bill</first-name>
  <last-name>Customer</last-name>
  <street>28 Red Hat Way</street>
  <city>Boston</city>
  <state>MA</state>
  <zip>02115</zip>
  <country>USA</country>
</customer>
```

**Figure 8.12. Customer data**

d. Click the Invoke button.

**Result:** The **Response Headers** area indicated that a record was created and lists the location as `http://localhost:8080/simple/rest-services/customers/1` as shown in Figure 8.13, “Customer added”.

![Response Headers](image)
3. **Test the GET option**

   a. Select the **GET** method in the operations drop-down list.

   b. We will retrieve the record created in the previous step. Enter the url for the record returned in the previous step: `http://localhost:8080/simple/rest-services/customers/1`

   c. Click the **Invoke** button.

**Result:** The **Response Headers** area indicates a [HTTP/1.1 200 OK] response and the customer data is retrieved and displayed in the **Response Body** area as shown in Figure 8.14, “GET response”.

### Response Body

```
<customer id="1">
  <first-name>Bill</first-name>
  <last-name>Customer</last-name>
  <street>28 Red Hat Way</street>
  <city>Boston</city>
  <state>MA</state>
  <zip>02115</zip>
  <country>USA</country>
</customer>
```

**Figure 8.14. GET response**
4. **Test the PUT option**

   a. Editing a record is achieved by using the **PUT** method. Select the **PUT** method in the operations drop-down list.

   b. Enter the url of the record to be edited: `http://localhost:8080/simple/rest-services/customers/1`

   c. Enter the data in the **Body Text** area. Replace the first-name with a different entry as in Figure 8.15, “Updated customer data”

```
<customer>
  <first-name>Terry</first-name>
  <last-name>Customer</last-name>
  <street>28 Red Hat Way</street>
  <city>Boston</city>
  <state>MA</state><zip>02115</zip>
  <country>USA</country>
</customer>
```

**Figure 8.15. Updated customer data**

   d. Ensure that the `content-type=application/xml` header is in the **Headers** list.

   e. Click the **Invoke** button.

**Result:** The **Response Headers** area indicates a No Response ([HTTP/1.1 204 No Content]) Figure 8.16, “Response header following PUT”.

```
[HTTP/1.1 204 No Content]
Date=[Mon, 01 Nov 2010 10:51:28 GMT]
Server=[Apache-Coyote/1.1]
X-Powered-By=[Servlet 2.5; JBoss-5.0/JBossWeb-2.1]
```

**Figure 8.16. Response header following PUT**

In this instance, the console does not indicate an update was performed, however, the console may provide useful information following an operation.
5. **Check the updated data with a GET**

   Perform a GET operation by following the steps in *Step 3*.

   **Result:** The **Response Body** area displays the updated data.

   ```xml
   <customer id="1">
     <first-name>Terry</first-name>
     <last-name>Customer</last-name>
     <street>28 Red Hat Way</street>
     <city>Boston</city>
     <state>MA</state>
     <zip>02115</zip>
     <country>USA</country>
   </customer>
   ``

   **Figure 8.17. Customer data updated**

6. **Test the DELETE option**

   a. Deleting a record is a similar process to posting. Select the **DELETE** method in the operations drop-down list.

   b. Enter the url of the record to be deleted: `http://localhost:8080/simple/rest-services/customers/1`

   c. Click the **Invoke** button.

   **Result:** The **Response Headers** area indicates a No Response ([HTTP/1.1 204 No Content]) as was the case for the PUT operation in *Figure 8.16, “Response header following PUT”*.

   Once again, the console does not indicate an update was performed, however, the console may provide useful information following an operation.

7. **Check the DELETE operation with a GET**

   Perform a GET operation by following the steps in *Step 3*.

   **Result:** The **Response Body** area returns an error report indicating that the requested resource () is not available and the **Response Headers** area returns a [HTTP/1.1 404 Not Found].
Figure 8.18. Customer data deleted

The response header and body messages indicate that the data was successfully deleted.