JBoss RESTful Web Services
Runtime and Tools support
Overview

JBoss RESTful Web Services is a framework developed as a part of the JBoss Application Server. It implements the JAX-RS specifications. 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.

JBoss RESTful Web Services integrates with most current JBoss Application Server releases as well as earlier ones, that did implement the J2EE 1.4 specifications.

RESTful Web Services tooling works with JBossWS Runtime and allows you to create, deploy and run RESTful Web Services.

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

1.1. Key Features of JBoss RESTful Web Services

Table 1.1. Key Functionality

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<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>JBoss AS</td>
<td>JBoss Application Server 5 (JavaEE 5 compliant) web service stack.</td>
</tr>
</tbody>
</table>
Sample Web Service wizards

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

- 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 2.1. 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.
Chapter 2. Sample Web Service...

Figure 2.1. Dynamic Web Project Attributes

3. **Define the Project Attributes**

Define the Dynamic Web Project attributes according to the options displayed in Table 2.1, “New Dynamic Web Project”
### Table 2.1. 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 2.2, “New Dynamic Project - Dynamic web module version”.</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 2.3, “New Dynamic Project - Configuration”.</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>
Chapter 2. Sample Web Service...

Table 2.2. 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 2.3. 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.

![New Dynamic Web Project - Web Module](image)

**Figure 2.2. New Dynamic Web Project - Web Module**

7. **Enter the web module settings**

Define the settings as listed in **Table 2.4, “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 2.4. 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 2. Sample Web Service...

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate web.xml deployment descriptor</td>
<td>no</td>
<td>Check this box to generate a deployment descriptor for the project.</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.

2.1. **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 2.2. Generate a sample RESTful 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 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.

![Generate a Sample RESTful Web Service](image)

**Project and Web Service Name**

RESTEasy jars not found in the project-associated runtime. Verify RE is installed and try again or check 'Add RESTEasy jars' below if enabled.

- **Dynamic Web Project:** testRESTful
- **Web Service**
  - **Name:** myRESTApplication
  - **Update web.xml**
  - **Add RESTEasy jars from root runtime directory**
- **Sample Web Service Class**
  - **Package:** org.jboss.samples.rs.webservices
  - **Class:** HelloWorldResource
  - **Application Class Name:** myRESTApplication

**Figure 2.3. Generate a Sample RESTful Web Service - Project and Web Service Name**

Due to the nature in which JBoss Application Server 7 and JBoss Enterprise Application Server 5 handle JAX-RS support, the wizard can now be completed without the need for RESTEasy JARs in the project classpath or associated project runtime. The JARs are necessary for JBoss SOA-P servers.

3. **Define the service attributes**

Define the project, web service, package and class names according to the options displayed in Table 2.5, “Project and Web Service Name”
## Table 2.5. 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></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>
<tr>
<td></td>
<td>Update web.xml</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></td>
<td>Add RESTEasy Jars from root runtime directory</td>
<td>no</td>
<td>Check this box to add RESTEasy JARs to the project.</td>
<td>This option allows you to add RESTEasy JARs to the project if they appear in the root runtime directory but are not installed in the runtime. While this is not required, it will assist when working with JBoss Application Server 5 and JBoss Enterprise Application Platform 5 web service projects.</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>
</tbody>
</table>
### Dialog group Field Manda Instruction Description

| Application Class Name | Enter the name of the Application Class Name. | 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. |

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 2.4. Application Class - MyRESTApplication.java](image)

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

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

   - 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 2.6. web.xml

The url for sending GET requests is therefore `http://localhost:8080/ProjectName/` 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.

3.1. 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 3.1. 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].

![web.xml](image)

**Figure 3.1. web.xml**

The url for sending GET requests can be formed from `http://localhost:8080/ProjectName/[1]/[2]` or, `http://localhost:8080/simple/rest-services/customers..`
Procedure 3.2. 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.

   ![Export - WAR Export dialog](image)

2. Complete the export dialog

   Define the WAR file attributes as described in Table 3.1, “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</td>
</tr>
<tr>
<td>Field</td>
<td>Mandatory</td>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Project Explorer</td>
<td></td>
<td></td>
<td><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> directory of the target server profile.</td>
</tr>
<tr>
<td>Optimize for a specific server runtime</td>
<td>no</td>
<td>Select this box to optimize the <strong>WAR</strong> file for deployment to the targeted runtime.</td>
<td>The list of available runtimes will be those configured during the project set-up or by selecting <strong>File</strong> → <strong>New</strong> → <strong>Server</strong>.</td>
</tr>
</tbody>
</table>

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**.
Chapter 4.

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 4.1. 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 4.1. Web Service Test View

The main components of the Web Service Tester View are:

- WSDL path/button bar (Table 4.1, “WSDL path/button bar”)
- Request details panel (Table 4.2, “Request details panel”)
- Response details panel (Table 4.3, “Response details panel”)
Table 4.1. 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 JAX-WS or any other option to test a JAX-RS service using HTTP request methods (PUT, GET, POST, DELETE or OPTIONS).</td>
</tr>
<tr>
<td>Toolbar button - Get From WSDL</td>
<td>Click this button to display the Select WSDL dialog. Enter the URL, File system location or Eclipse Workspace location of the WSDL file. Given a valid file, the dialog will allow selection of the Port and Operation to test. Once selected, the request details will be displayed in the Request Details 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. Response details will be displayed in the Response Details panel.</td>
</tr>
</tbody>
</table>

Table 4.2. 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 (Add) one or more name=value 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 name=value pairs by clicking the Add 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 4.3. 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 from the web service invocation can be displayed by clicking the Show Raw button. The output will be embedded in a html browser by clicking the Show in Browser</td>
</tr>
</tbody>
</table>
The following sections describe testing JAX-RS web services, including the necessary preliminary steps.

### 4.1. Preliminaries

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

**Procedure 4.2. Testing a 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 4.2. Web Service Test View](image)

  **Figure 4.2. Web Service Test View**

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

  c. The server has been started with **run.sh -c <profile>**
4.2. 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 4.1, “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.

4.2.1. RestfulSample project

Procedure 4.3. 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 4.3, “JAX-RS Response Header Text”.

   ![Response Details](image)

   **Response Headers**

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

   ![Figure 4.3. JAX-RS Response Header Text](image)
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 4.4, “JAX-RS Response Body Text”.*

---

**Response Body**

Hello World!

---

*Figure 4.4. JAX-RS Response Body Text*

### 4.2.2. RESTEasy sample project

**Procedure 4.4. Testing a JAX-RS web service- POST and GET requests**

1. a. **Query the available options**

   Following the preliminary steps described in *Testing a 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 4.5, “JAX-RS RESTEasy project Body Text”.*
Figure 4.5. JAX-RS RESTEasy project Body Text

2. Test the POST option

   a. Select **POST** method in 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 4.6, “content-type header”**.

   c. **Enter customer details**

      Enter the customer details in the **Body Text** area as displayed in **Figure 4.7, “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 4.8, “Customer added”.**

**Response Headers**

- [HTTP/1.1 201 Created]
- Date=[Wed, 27 Oct 2010 12:29:00 GMT]
- Content-Length=[0]
- Location=[http://localhost:8080/simple/rest-services/customers/1]
- Server=[Apache-Coyote/1.1]
- X-Powered-By=[Servlet 2.5; JBoss-5.0/JBossWeb-2.1]

**Figure 4.8. Customer added**

The console also indicates the successful creation of the customer: **10:44:33, 846 INFO [STDOUT] Created customer 1**

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 4.9, “GET response”.

- Response Body

```xml
<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 4.9. 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 4.10, “Updated customer data”

- Body Text

```xml
<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 4.10. 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 4.11, “Response header following PUT”.

Response Headers

[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 4.11. 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.

Response Body

<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 4.12. 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 4.11, “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].

   □ **Response Headers**

      ![HTTP/1.1 404 Not Found]

      Date=[Mon, 01 Nov 2010 11:23:55 GMT]
      Content-Length=[942]
      Content-Type=[text/html;charset=utf-8]
      Server=[Apache-Coyote/1.1]

   **Figure 4.13. Customer data deleted**

   The response header and body messages indicate that the data was successfully deleted.
**JAX-RS Validation**

JAX-RS validation is enabled by default. Validation allows your project to be checked for errors. If an error is discovered a **Problems** tab will appear in the bottom section of your workbench, outlining the errors found.

If you wish to turn off JAX-RS Validation, you can do so by first navigating to **Window → Preferences → Validation**. In the **Validator** section of the dialog, deselect the checkboxes for **JAX-RS Metamodel Validator** and click the **Apply** button, followed by **OK**.
## Chapter 5. JAX-RS Validation

### Figure 5.1. Validator preferences

![Validator preferences](image)

<table>
<thead>
<tr>
<th>Validator</th>
<th>Manual</th>
<th>Build</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facelet HTML Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
<tr>
<td>HTML Syntax Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
<tr>
<td>JAXB Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
<tr>
<td>JAX-RS Metamodel Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
<tr>
<td>JBoss Tools Project Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
<tr>
<td>JPA Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
<tr>
<td>JSF Application Configuration Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
<tr>
<td>JSF View Validator</td>
<td>☑</td>
<td>☑</td>
<td>...</td>
</tr>
</tbody>
</table>

- Allow projects to override these preference settings
- Suspend all validators
- Save all modified resources automatically prior to validating
- Show a confirmation dialog when performing manual validations

The selected validators will run when validation is performed.