* [Home](http://www.vogella.com/)
* [Tutorials](http://www.vogella.com/tutorials.html)
* [Training](http://www.vogella.com/training/overview.html)
* [Books](http://www.vogella.com/books.html)
* [Connect](http://www.vogella.com/social.html)

[](http://www.vogella.com/)

[](http://www.vogella.com/)

*[](http://www.vogella.com/about.html)*

*by Lars Vogel*

**Tutorial**

[](http://www.vogella.com/training/overview.html)[**Android Training**](http://www.vogella.com/training/android.html) [**Eclipse Training**](http://www.vogella.com/training/eclipsercp.html)

[](http://www.twitter.com/vogella)

[](https://profiles.google.com/104044918586174763681)

**REST with Java (JAX-RS) using Jersey - Tutorial**

**Lars Vogel**

Version 1.6

Copyright © 2009, 2010, 2011, 2012 Lars Vogel

04.06.2012

|  |  |  |
| --- | --- | --- |
| **Revision History** | | |
| Revision 0.1 | 11.05.2009 | Lars Vogel |
| Created Article | | |
| Revision 0.2 - 1.6 | 10.10.2009 - 04.06.2012 | Lars Vogel |
| bug fixes and enhancements | | |

**RESTful Webservices with Java (Jersey / JAX-RS)**

This tutorial explains how to develop RESTful web services in Java with the JAX-RS reference implementation Jersey.

In this tutorial Eclipse 3.6 (Helios), Java 1.6, Tomcat 6.0 and JAX-RS 1.1. (Jersey 1.5) is used.

**Table of Contents**

[**1. REST - Representational State Transfer**](http://www.vogella.com/articles/REST/article.html#rest)

[**1.1. Overview**](http://www.vogella.com/articles/REST/article.html#rest_overview)

[**1.2. HTTP methods**](http://www.vogella.com/articles/REST/article.html#rest_httpmethods)

[**1.3. RESTFul webservices**](http://www.vogella.com/articles/REST/article.html#rest_webservices)

[**1.4. Java, REST and Jersey**](http://www.vogella.com/articles/REST/article.html#rest_jaxrs)

[**2. Installation**](http://www.vogella.com/articles/REST/article.html#installation)

[**2.1. Jersey**](http://www.vogella.com/articles/REST/article.html#installation_jersey)

[**2.2. Eclipse and Tomcat**](http://www.vogella.com/articles/REST/article.html#installation_eclipse)

[**3. Create your first RESTful Webservice**](http://www.vogella.com/articles/REST/article.html#first)

[**3.1. Create project**](http://www.vogella.com/articles/REST/article.html#first_project)

[**3.2. Java Class**](http://www.vogella.com/articles/REST/article.html#first_class)

[**3.3. Define Jersey Servlet dispatcher**](http://www.vogella.com/articles/REST/article.html#first_servletdispatcher)

[**3.4. Run your rest service**](http://www.vogella.com/articles/REST/article.html#first_run)

[**3.5. Create a client**](http://www.vogella.com/articles/REST/article.html#first_client)

[**4. Restful webservices and JAXB**](http://www.vogella.com/articles/REST/article.html#jaxb)

[**4.1. Create project**](http://www.vogella.com/articles/REST/article.html#jaxb_project)

[**4.2. Create a client**](http://www.vogella.com/articles/REST/article.html#jaxb_client)

[**5. CRUD RESTful webservice**](http://www.vogella.com/articles/REST/article.html#crud)

[**5.1. Project**](http://www.vogella.com/articles/REST/article.html#crud_project)

[**5.2. Create a simple HTML form**](http://www.vogella.com/articles/REST/article.html#crud_webpage)

[**5.3. Rest Service**](http://www.vogella.com/articles/REST/article.html#crud_class)

[**5.4. Run**](http://www.vogella.com/articles/REST/article.html#crud_run)

[**5.5. Create a client**](http://www.vogella.com/articles/REST/article.html#crud_client)

[**5.6. Using the rest service via HTML page**](http://www.vogella.com/articles/REST/article.html#crud_html)

[**5.7. Using the rest service via X**](http://www.vogella.com/articles/REST/article.html#crud_lib)

[**6. Thank you**](http://www.vogella.com/articles/REST/article.html#thankyou)

[**7. Questions and Discussion**](http://www.vogella.com/articles/REST/article.html#questions)

[**8. Links and Literature**](http://www.vogella.com/articles/REST/article.html#resources)

[**8.1. Source Code**](http://www.vogella.com/articles/REST/article.html#sourcecode)

[**8.2. Rest Resources**](http://www.vogella.com/articles/REST/article.html#resources_rest)

[**8.3. vogella Resources**](http://www.vogella.com/articles/REST/article.html#resources_general)

**1. REST - Representational State Transfer**

**1.1. Overview**

REST is an architectural style which is based on web-standards and the HTTP protocol. REST was first described by Roy Fielding in 2000.

In a REST based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods. In an REST architecture you typically have a REST server which provides access to the resources and a REST client which accesses and modify the REST resources. Every resource should support the HTTP common operations. Resources are identified by global ID's (which are typically URIs).

REST allows that resources have different representations, e.g. text, xml, json etc. The rest client can ask for specific representation via the HTTP protocol (Content Negotiation).

**1.2. HTTP methods**

The HTTP standards methods which are typical used in REST are PUT, GET, POST, DELETE.

* GET defines a reading access of the resource without side-effects. The resource is never changed via a GET request, e.g. the request has no side effects (idempotent).
* PUT creates a new resource, must also be idempotent.
* DELETE removes the resources. The operations are idempotent, they can get repeated without leading to different results.
* POST updates an existing resource or creates a new resource.

**1.3. RESTFul webservices**

A RESTFul webservices is based on the HTTP methods and the concept of REST. It typically defines the base URI for the services, the MIME-types its supports (XML, Text, JSON, user-defined,..) and the set of operations (POST, GET, PUT, DELETE) which are supported. JAX-RS supports the creation of XML and JSON via [**JAXB**](http://www.vogella.com/articles/JAXB/article.html).

**1.4. Java, REST and Jersey**

Java defines standard REST support via JAX-RS (The Java API for RESTful Web Services) in [**JSR 311**](http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html).JAX-RS uses annotations to define the REST relevance of classes.

Via your "web.xml" you will register a servlet provided by Jersey and also define the path under which your REST web application will be available. The base URL of this servlet is:

http:*//your\_domain:port/display-name/url-pattern/path\_from\_rest\_class*

This servlet which analyze the incoming HTTP request and select the correct class and method to respond to this request. This selection is based on annotation in the class and methods.

The most important annotations in JAX-RS are the following.

**Table 1. Sample Table**

| **Annotation** | **Description** |
| --- | --- |
| @PATH(your\_path) | Sets the path to base URL + /your\_path. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file. |
| @POST | Indicates that the following method will answer to a HTTP POST request |
| @GET | Indicates that the following method will answer to a HTTP GET request |
| @PUT | Indicates that the following method will answer to a HTTP PUT request |
| @DELETE | Indicates that the following method will answer to a HTTP DELETE request |
| @Produces(MediaType.TEXT\_PLAIN [, more-types ]) | @Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/plain") is produced. Other examples would be "application/xml" or "application/json". |
| @Consumes(type [, more-types ]) | @Consumes defines which MIME type is consumed by this method. |
| @PathParam | Used to inject values from the URL into a method parameter. This way you inject for example the ID of a resource into the method to get the correct object. |

The complete path to a resource is therefore based on the base URL and the @PATh annotation in your class.

http:*//your\_domain:port/display-name/url-pattern/path\_from\_rest\_class*

[**Jersey**](https://jersey.dev.java.net/) is the reference implementation for this specification. Jersey contains basically a REST server and a REST client. The core client is mainly available for testing and provides a library to communicate with the server.

A REST web application consists out of data classes (resources) and services. These two types are typically maintained in different packages as the Jersey servlet will be instructed via the "web.xml"to scan certain packages for data classes.

**2. Installation**

**2.1. Jersey**

Download Jersey from [**Jersey Homepage**](https://jersey.dev.java.net/) . As of the time of writing the file is called ""A zip of Jersey containing the Jersey jars, core dependencies (it does not provide dependencies for third party jars beyond the those for JSON support) and JavaDoc." Download this zip; it contains the jar files required for the REST functionality.

**2.2. Eclipse and Tomcat**

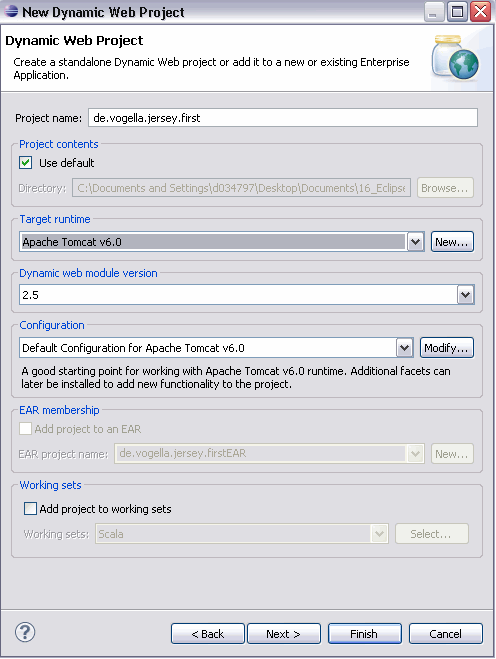
This tutorial is using [**Tomcat**](http://www.vogella.com/articles/ApacheTomcat/article.html) as servlet container and Eclipse WTP as a development environment. Please see [**Eclipse WTP**](http://www.vogella.com/articles/EclipseWTP/article.html) and [**Apache Tomcat**](http://www.vogella.com/articles/ApacheTomcat/article.html) for instructions on how to install and use Eclipse WTP and Apache Tomcat.

Alternative you could also use the [**Google App Engine**](http://www.vogella.com/articles/GoogleAppEngineJava/article.html) for running the server part of the following REST examples. If you use the Google App Engine you don't have to setup Tomcat. If you are using GAE/J you have to create App Engine projects instead of "Dynamic Web Project".

**3. Create your first RESTful Webservice**

**3.1. Create project**

Create a new "Dynamic Web Project" "de.vogella.jersey.first". See [**Eclipse WTP development**](http://www.vogella.com/articles/EclipseWTP/article.html) for how to create a new dynamic web project.



Copy all jars from your Jersey download into the folder "WEB-INF/lib".

**3.2. Java Class**

Create the following class.

**package** de.vogella.jersey.first;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.MediaType;

*// POJO, no interface no extends*

*// The class registers its methods for the HTTP GET request using the @GET annotation.*

*// Using the @Produces annotation, it defines that it can deliver several MIME types,*

*// text, XML and HTML.*

*// The browser requests per default the HTML MIME type.*

*//Sets the path to base URL + /hello*

*@Path("/hello")*

**public** **class** Hello {

*// This method is called if TEXT\_PLAIN is request*

*@GET*

*@Produces(MediaType.TEXT\_PLAIN)*

**public** String sayPlainTextHello() {

**return** "Hello Jersey";

}

*// This method is called if XML is request*

*@GET*

*@Produces(MediaType.TEXT\_XML)*

**public** String sayXMLHello() {

**return** "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";

}

*// This method is called if HTML is request*

*@GET*

*@Produces(MediaType.TEXT\_HTML)*

**public** String sayHtmlHello() {

**return** "<html> " + "<title>" + "Hello Jersey" + "</title>"

+ "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";

}

}

This class register itself as a get resource via the @GET annotation. Via the @Produces annotation it defines that it delivers two MIME types, "text" and "HTML" code. It also defines via the "Path" annotation that its service should be available under the URL "hello".

The browser will always request the html MIME type. To see the text version you can use tool like [**curl**](http://www.vogella.com/articles/Ubuntu/article.html#network_curl) .

**3.3. Define Jersey Servlet dispatcher**

You need to register Jersey as the servlet dispatcher for REST requests. Open the file "web.xml" and modify the file to the following.

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">

<display-name>de.vogella.jersey.first</display-name>

<servlet>

<servlet-name>Jersey REST Service</servlet-name>

<servlet-**class**>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-**class**>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>de.vogella.jersey.first</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey REST Service</servlet-name>

<url-pattern>/rest*/\*</url-pattern>*

*</servlet-mapping>*

*</web-app>*

The parameter "com.sun.jersey.config.property.package" defines in which package jersey will look for the web service classes. This property must point to your resources classes. The URL pattern defines part of the base URL your application will be placed.

**3.4. Run your rest service**

Run you web application in Eclipse. See [**Eclipse WTP**](http://www.vogella.com/articles/EclipseWTP/article.html) for details on how to run dynamic web applications.

Test your REST service under: "http://localhost:8080/de.vogella.jersey.first/rest/hello". This name is derived from the "display-name" defined in the "web.xml" file, augmented with the servlet-mapping url-pattern and the "hello" @Path annotation from your class file. You should get the message "Hello Jersey".

The browser requests the HTML representation of your resource. In the next chapter we are going to write a client which will read the XML representation.

**3.5. Create a client**

Jersey contains a REST client library which can be used for testing or to build a real client in Java. Alternative you could use [**Apache HttpClient**](http://www.vogella.com/articles/ApacheHttpClient/article.html) to create a client.

Create a new Java "de.vogella.jersey.first.client" and add the jersey jars to the project and the project build path. Create the following test class.

**package** de.vogella.jersey.first.client;

**import** java.net.URI;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.UriBuilder;

**import** com.sun.jersey.api.client.Client;

**import** com.sun.jersey.api.client.ClientResponse;

**import** com.sun.jersey.api.client.WebResource;

**import** com.sun.jersey.api.client.config.ClientConfig;

**import** com.sun.jersey.api.client.config.DefaultClientConfig;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

ClientConfig config = **new** DefaultClientConfig();

Client client = Client.create(config);

WebResource service = client.resource(getBaseURI());

*// Fluent interfaces*

System.out.println(service.path("rest").path("hello").accept(MediaType.TEXT\_PLAIN).get(ClientResponse.**class**).toString());

*// Get plain text*

System.out.println(service.path("rest").path("hello").accept(MediaType.TEXT\_PLAIN).get(String.**class**));

*// Get XML*

System.out.println(service.path("rest").path("hello").accept(MediaType.TEXT\_XML).get(String.**class**));

*// The HTML*

System.out.println(service.path("rest").path("hello").accept(MediaType.TEXT\_HTML).get(String.**class**));

}

**private** **static** URI getBaseURI() {

**return** UriBuilder.fromUri("http://localhost:8080/de.vogella.jersey.first").build();

}

}

**4. Restful webservices and JAXB**

JAX-RS supports the automatic creation of XML and JSON via JAXB. For an introduction into XML please see [**Java and XML - Tutorial**](http://www.vogella.com/articles/JavaXML/article.html) . For an introduction into JAXB please see [**JAXB**](http://www.vogella.com/articles/JAXB/article.html). You can continue this tutorial without reading these tutorials but they contain more background information.

**4.1. Create project**

Create a new "Dynamic Web Project" "de.vogella.jersey.jaxb" and copy all jersey jars into the folder "WEB-INF/lib".

Create your domain class.

**package** de.vogella.jersey.jaxb.model;

**import** javax.xml.bind.annotation.XmlRootElement;

*@XmlRootElement*

*// JAX-RS supports an automatic mapping from JAXB annotated class to XML and JSON*

*// Isn't that cool?*

**public** **class** Todo {

**private** String summary;

**private** String description;

**public** String getSummary() {

**return** summary;

}

**public** **void** setSummary(String summary) {

**this**.summary = summary;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

}

Create the following resource class. This class simply return an instance of the Todo class.

**package** de.vogella.jersey.jaxb;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.MediaType;

**import** de.vogella.jersey.jaxb.model.Todo;

*@Path("/todo")*

**public** **class** TodoResource {

*// This method is called if XMLis request*

*@GET*

*@Produces({ MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON })*

**public** Todo getXML() {

Todo todo = **new** Todo();

todo.setSummary("This is my first todo");

todo.setDescription("This is my first todo");

**return** todo;

}

*// This can be used to test the integration with the browser*

*@GET*

*@Produces({ MediaType.TEXT\_XML })*

**public** Todo getHTML() {

Todo todo = **new** Todo();

todo.setSummary("This is my first todo");

todo.setDescription("This is my first todo");

**return** todo;

}

}

Change "web.xml" to the following.

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

id="WebApp\_ID" version="2.5">

<display-name>de.vogella.jersey.jaxb</display-name>

<servlet>

<servlet-name>Jersey REST Service</servlet-name>

<servlet-**class**>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-**class**>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>de.vogella.jersey.jaxb</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey REST Service</servlet-name>

<url-pattern>/rest*/\*</url-pattern>*

*</servlet-mapping>*

*</web-app>*

Run you web application in Eclipse and validate that you can access your service. Your application should be available under "http://localhost:8080/de.vogella.jersey.jaxb/rest/todo".

**4.2. Create a client**

Create a new Java "de.vogella.jersey.jaxb.client" and add the jersey jars to the project and the project build path. Create the following test class.

**package** de.vogella.jersey.jaxb.client;

**import** java.net.URI;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.UriBuilder;

**import** com.sun.jersey.api.client.Client;

**import** com.sun.jersey.api.client.WebResource;

**import** com.sun.jersey.api.client.config.ClientConfig;

**import** com.sun.jersey.api.client.config.DefaultClientConfig;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

ClientConfig config = **new** DefaultClientConfig();

Client client = Client.create(config);

WebResource service = client.resource(getBaseURI());

*// Get XML*

System.out.println(service.path("rest").path("todo").accept(MediaType.TEXT\_XML).get(String.**class**));

*// Get XML for application*

System.out.println(service.path("rest").path("todo").accept(MediaType.APPLICATION\_JSON).get(String.**class**));

*// Get JSON for application*

System.out.println(service.path("rest").path("todo").accept(MediaType.APPLICATION\_XML).get(String.**class**));

}

**private** **static** URI getBaseURI() {

**return** UriBuilder.fromUri("http://localhost:8080/de.vogella.jersey.jaxb").build();

}

}

**5. CRUD RESTful webservice**

This section creates a CRUD (Create, Read, Update, Delete) restful web service. It will allow to maintain a list of todos in your web application via HTTP calls.

**5.1. Project**

Create a new dynamic project "de.vogella.jersey.todo" and add the jersey libs. Modify "web.xml" to the following.

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">

<display-name>de.vogella.jersey.todo</display-name>

<servlet>

<servlet-name>Jersey REST Service</servlet-name>

<servlet-**class**>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-**class**>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>de.vogella.jersey.todo.resources</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey REST Service</servlet-name>

<url-pattern>/rest*/\*</url-pattern>*

*</servlet-mapping>*

*</web-app>*

Create the following data model and a [**Singleton**](http://www.vogella.com/articles/DesignPatternSingleton/article.html) which serves as the data provider for the model. We use the implementation based on an enumeration. Please see the link for details. The Todo class is annotated with a JAXB annotation. See [**Java and XML**](http://www.vogella.com/articles/JavaXML/article.html) to learn about JAXB.

**package** de.vogella.jersey.todo.model;

**import** javax.xml.bind.annotation.XmlRootElement;

*@XmlRootElement*

**public** **class** Todo {

**private** String id;

**private** String summary;

**private** String description;

**public** Todo(){

}

**public** Todo (String id, String summary){

**this**.id = id;

**this**.summary = summary;

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getSummary() {

**return** summary;

}

**public** **void** setSummary(String summary) {

**this**.summary = summary;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

}

**package** de.vogella.jersey.todo.dao;

**import** java.util.HashMap;

**import** java.util.Map;

**import** de.vogella.jersey.todo.model.Todo;

**public** enum TodoDao {

instance;

**private** Map<String, Todo> contentProvider = **new** HashMap<String, Todo>();

**private** TodoDao() {

Todo todo = **new** Todo("1", "Learn REST");

todo.setDescription("Read http://www.vogella.com/articles/REST/article.html");

contentProvider.put("1", todo);

todo = **new** Todo("2", "Do something");

todo.setDescription("Read complete http://www.vogella.com");

contentProvider.put("2", todo);

}

**public** Map<String, Todo> getModel(){

**return** contentProvider;

}

}

**5.2. Create a simple HTML form**

The rest service can be used via HTML forms. The following HTML form will allow to post new data to the service. Create the following page "create\_todo.html" in the folder "WEB-INF".

<!DOCTYPE html>

<html>

<head>

<title>Form to create a **new** resource</title>

</head>

<body>

<form action="../de.vogella.jersey.todo/rest/todos" method="POST">

<label **for**="id">ID</label>

<input name="id" />

<br/>

<label **for**="summary">Summary</label>

<input name="summary" />

<br/>

Description:

<TEXTAREA NAME="description" COLS=40 ROWS=6></TEXTAREA>

<br/>

<input type="submit" value="Submit" />

</form>

</body>

</html>

**5.3. Rest Service**

Create the following classes which will be used as REST resources.

**package** de.vogella.jersey.todo.resources;

**import** javax.ws.rs.Consumes;

**import** javax.ws.rs.DELETE;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.PUT;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.Context;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.Request;

**import** javax.ws.rs.core.Response;

**import** javax.ws.rs.core.UriInfo;

**import** javax.xml.bind.JAXBElement;

**import** de.vogella.jersey.todo.dao.TodoDao;

**import** de.vogella.jersey.todo.model.Todo;

**public** **class** TodoResource {

*@Context*

UriInfo uriInfo;

*@Context*

Request request;

String id;

**public** TodoResource(UriInfo uriInfo, Request request, String id) {

**this**.uriInfo = uriInfo;

**this**.request = request;

**this**.id = id;

}

*//Application integration*

*@GET*

*@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})*

**public** Todo getTodo() {

Todo todo = TodoDao.instance.getModel().get(id);

**if**(todo==null)

**throw** **new** RuntimeException("Get: Todo with " + id + " not found");

**return** todo;

}

*// For the browser*

*@GET*

*@Produces(MediaType.TEXT\_XML)*

**public** Todo getTodoHTML() {

Todo todo = TodoDao.instance.getModel().get(id);

**if**(todo==null)

**throw** **new** RuntimeException("Get: Todo with " + id + " not found");

**return** todo;

}

*@PUT*

*@Consumes(MediaType.APPLICATION\_XML)*

**public** Response putTodo(JAXBElement<Todo> todo) {

Todo c = todo.getValue();

**return** putAndGetResponse(c);

}

*@DELETE*

**public** **void** deleteTodo() {

Todo c = TodoDao.instance.getModel().remove(id);

**if**(c==null)

**throw** **new** RuntimeException("Delete: Todo with " + id + " not found");

}

**private** Response putAndGetResponse(Todo todo) {

Response res;

**if**(TodoDao.instance.getModel().containsKey(todo.getId())) {

res = Response.noContent().build();

} **else** {

res = Response.created(uriInfo.getAbsolutePath()).build();

}

TodoDao.instance.getModel().put(todo.getId(), todo);

**return** res;

}

}

**package** de.vogella.jersey.todo.resources;

**import** java.io.IOException;

**import** java.net.URI;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.ws.rs.Consumes;

**import** javax.ws.rs.FormParam;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.POST;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.PathParam;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.Context;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.Request;

**import** javax.ws.rs.core.Response;

**import** javax.ws.rs.core.UriInfo;

**import** de.vogella.jersey.todo.dao.TodoDao;

**import** de.vogella.jersey.todo.model.Todo;

*// Will map the resource to the URL todos*

*@Path("/todos")*

**public** **class** TodosResource {

*// Allows to insert contextual objects into the class,*

*// e.g. ServletContext, Request, Response, UriInfo*

*@Context*

UriInfo uriInfo;

*@Context*

Request request;

*// Return the list of todos to the user in the browser*

*@GET*

*@Produces(MediaType.TEXT\_XML)*

**public** List<Todo> getTodosBrowser() {

List<Todo> todos = **new** ArrayList<Todo>();

todos.addAll(TodoDao.instance.getModel().values());

**return** todos;

}

*// Return the list of todos for applications*

*@GET*

*@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})*

**public** List<Todo> getTodos() {

List<Todo> todos = **new** ArrayList<Todo>();

todos.addAll(TodoDao.instance.getModel().values());

**return** todos;

}

*// retuns the number of todos*

*// Use http://localhost:8080/de.vogella.jersey.todo/rest/todos/count*

*// to get the total number of records*

*@GET*

*@Path("count")*

*@Produces(MediaType.TEXT\_PLAIN)*

**public** String getCount() {

**int** count = TodoDao.instance.getModel().size();

**return** String.valueOf(count);

}

*@POST*

*@Produces(MediaType.TEXT\_HTML)*

*@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)*

**public** **void** newTodo(*@FormParam("id")* String id,

*@FormParam("summary")* String summary,

*@FormParam("description")* String description,

*@Context* HttpServletResponse servletResponse) **throws** IOException {

Todo todo = **new** Todo(id,summary);

**if** (description!=null){

todo.setDescription(description);

}

TodoDao.instance.getModel().put(id, todo);

servletResponse.sendRedirect("../create\_todo.html");

}

*// Defines that the next path parameter after todos is*

*// treated as a parameter and passed to the TodoResources*

*// Allows to type http://localhost:8080/de.vogella.jersey.todo/rest/todos/1*

*// 1 will be treaded as parameter todo and passed to TodoResource*

*@Path("{todo}")*

**public** TodoResource getTodo(*@PathParam("todo")* String id) {

**return** **new** TodoResource(uriInfo, request, id);

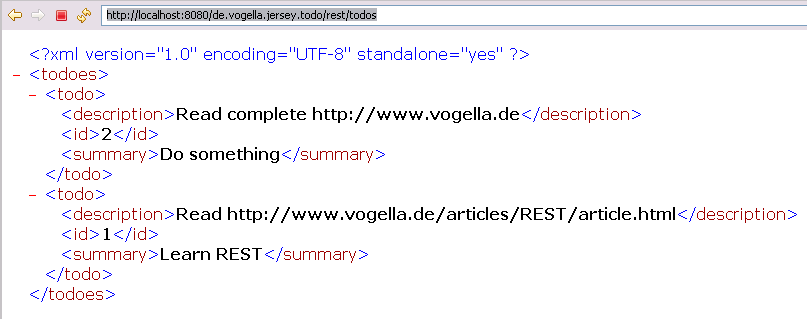
}

}

This TodosResource uses "@PathParam" annotation to use the parameter "id" to forward the request to the class "TodoResource".

**5.4. Run**

Run you web application in Eclipse and test the availability of your REST service under: "http://localhost:8080/de.vogella.jersey.todo/rest/todos". You should see the XML representation of your Todo items.



To see the count of Todo items use "http://localhost:8080/de.vogella.jersey.todo/rest/todos/count" to see an exiting todo use "http://localhost:8080/de.vogella.jersey.todo/rest/todos/{id}", e.g. "http://localhost:8080/de.vogella.jersey.todo/rest/todos/1" to see the todo with ID 1. We currently have only todos with the id's 1 and 2, all other requests will result an HTTP error code.

Please note that with the browser you can only issue HTTP GET requests. The next chapter will use the jersey client libraries to issue get, post and delete.

**5.5. Create a client**

Create a Java project "de.vogella.jersey.todo.client". Create a folder "lib", place all jersey libs there and add then to your classpath. Create the following class.

**package** de.vogella.jersey.todo.client;

**import** java.net.URI;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.UriBuilder;

**import** com.sun.jersey.api.client.Client;

**import** com.sun.jersey.api.client.ClientResponse;

**import** com.sun.jersey.api.client.WebResource;

**import** com.sun.jersey.api.client.config.ClientConfig;

**import** com.sun.jersey.api.client.config.DefaultClientConfig;

**import** com.sun.jersey.api.representation.Form;

**import** de.vogella.jersey.todo.model.Todo;

**public** **class** Tester {

**public** **static** **void** main(String[] args) {

ClientConfig config = **new** DefaultClientConfig();

Client client = Client.create(config);

WebResource service = client.resource(getBaseURI());

*// Create one todo*

Todo todo = **new** Todo("3", "Blabla");

ClientResponse response = service.path("rest").path("todos").path(todo.getId()).accept(MediaType.APPLICATION\_XML).put(ClientResponse.**class**, todo);

*// Return code should be 201 == created resource*

System.out.println(response.getStatus());

*// Get the Todos*

System.out.println(service.path("rest").path("todos").accept(MediaType.TEXT\_XML).get(String.**class**));

*// Get XML for application*

System.out.println(service.path("rest").path("todos").accept(MediaType.APPLICATION\_JSON).get(String.**class**));

*// Get JSON for application*

System.out.println(service.path("rest").path("todos").accept(MediaType.APPLICATION\_XML).get(String.**class**));

*// Get the Todo with id 1*

System.out.println(service.path("rest").path("todos/1").accept(MediaType.APPLICATION\_XML).get(String.**class**));

*// get Todo with id 1*

service.path("rest").path("todos/1").delete();

*// Get the all todos, id 1 should be deleted*

System.out.println(service.path("rest").path("todos").accept(MediaType.APPLICATION\_XML).get(String.**class**));

*// Create a Todo*

Form form = **new** Form();

form.add("id", "4");

form.add("summary", "Demonstration of the client lib for forms");

response = service.path("rest").path("todos").type(MediaType.APPLICATION\_FORM\_URLENCODED)

.post(ClientResponse.**class**, form);

System.out.println("Form response " + response.getEntity(String.**class**));

*// Get the all todos, id 4 should be created*

System.out.println(service.path("rest").path("todos").accept(MediaType.APPLICATION\_XML).get(String.**class**));

}

**private** **static** URI getBaseURI() {

**return** UriBuilder.fromUri("http://localhost:8080/de.vogella.jersey.todo").build();

}

}

**5.6. Using the rest service via HTML page**

The above example contains a form which calls a post method of your rest service.

**5.7. Using the rest service via X**

Usually every programming language provide somewhere libraries for creating HTTP get, post, put and delete requests. For Java the project [**Apache HttpClient**](http://www.vogella.com/articles/ApacheHttpClient/article.html).

**6. Thank you**

Please help me to support this article:

|  |  |
| --- | --- |
| **[Flattr this](http://flattr.com/thing/49772/Tutorials-for-Eclipse-Java-Android-and-Webprogramming)** | Top of Form    Bottom of Form |

**7. Questions and Discussion**

Before posting questions, please see the [**vogella FAQ**](http://www.vogella.com/faq.html). If you have questions or find an error in this article please use the [**www.vogella.com Google Group**](http://groups.google.com/group/vogella). I have created a short list [**how to create good questions**](http://www.vogella.com/blog/2010/03/09/asking-community-questions/) which might also help you.

**8. Links and Literature**

**8.1. Source Code**

[**Source Code of Examples**](http://www.vogella.com/code/index.html)

**8.2. Rest Resources**

[**http://jersey.java.net/**](http://jersey.java.net/) Jersey Homepage

[**http://www.ibm.com/developerworks/library/wa-aj-tomcat/**](http://www.ibm.com/developerworks/library/wa-aj-tomcat/) IBM Article about Rest with Tomcat and Jersey

**8.3. vogella Resources**

[**Eclipse RCP Training**](http://www.vogella.com/training/eclipsercp.html) (German) Eclipse RCP Training with Lars Vogel

[**Android Tutorial**](http://www.vogella.com/articles/Android/article.html) Introduction to Android Programming

[**GWT Tutorial**](http://www.vogella.com/articles/GWT/article.html) Program in Java and compile to JavaScript and HTML

[**Eclipse RCP Tutorial**](http://www.vogella.com/articles/EclipseRCP/article.html) Create native applications in Java

[**JUnit Tutorial**](http://www.vogella.com/articles/JUnit/article.html) Test your application

[**Git Tutorial**](http://www.vogella.com/articles/Git/article.html) Put everything you have under distributed version control system