1. REST - Representational State Transfer

1.1. What is REST?

REST is an architectural style which is based on web-standards and the HTTP protocol. This style was initially 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 a REST based architecture you have a REST server which provides access to the resources. A REST client can access and modify the REST resources.

Every resource should support the HTTP common operations. Resources are identified by global IDs (which are typically URIs).

REST allows that resources have different representations, e.g., text, XML, JSON etc. The REST client can ask for a specific representation via the HTTP protocol (content negotiation).

1.2. HTTP methods

The *PUT*, *GET*, *POST* and *DELETE* methods are typical used in REST based architectures. The following table gives an explanation of these operations.

- 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. It 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 web services

A RESTful web services are based on HTTP methods and the concept of REST. A RESTful web service defines the base URI for the services, the supported MIME-types (XML, text, JSON, user-defined, ...). It also defines the set of operations (POST, GET, PUT, DELETE) which are supported.

2. Installation of Jersey

2.1. Use Gradle

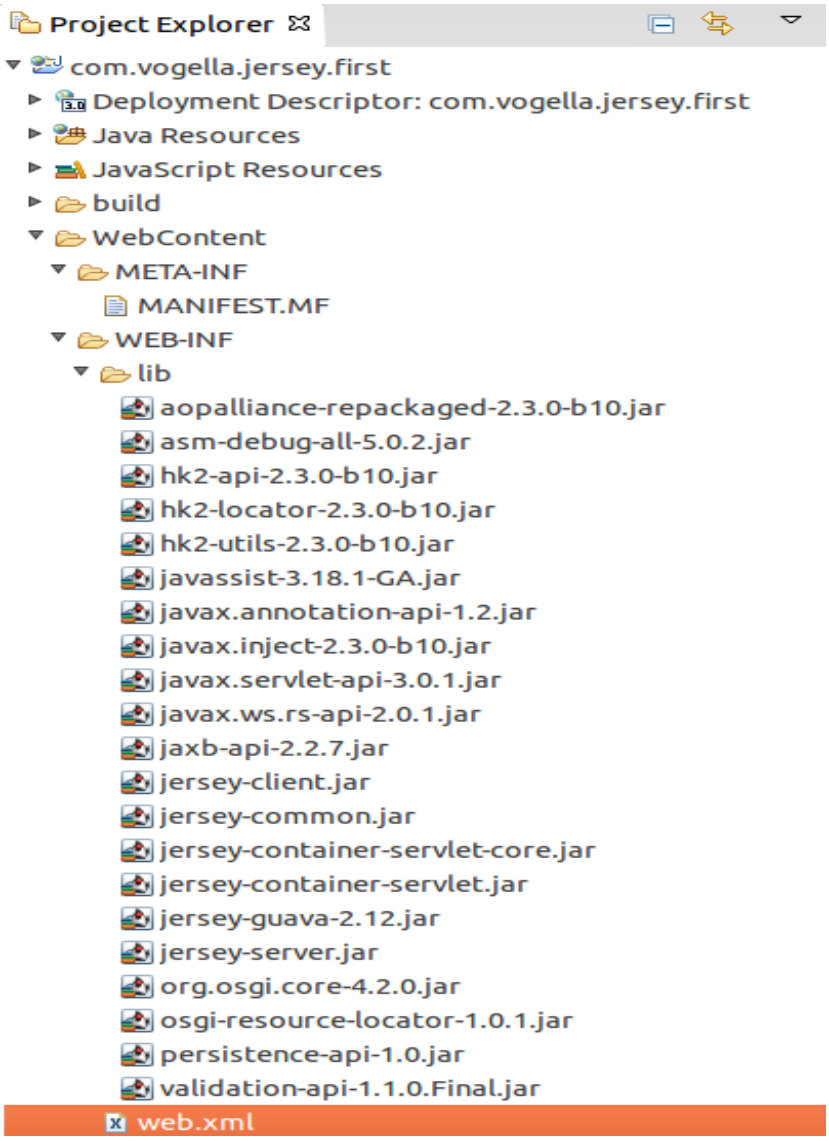
```
compile 'org.glassfish.jersey.containers:jersey-container-servlet:2.25.1'
```

2.2. Manual setup of Jersey libraries in an Eclipse project

Download the Jersey distribution as zip file from the Jersey download site.

The zip contains the Jersey implementation JAR and its core dependencies. It does not provide dependencies for third party JARs beyond those for JSON support and JavaDoc.

Copy all JARs from your Jersey download into the `WEB-INF/lib` folder.



3. Web container

For this tutorial you can use any web container, for example Tomcat or the Google App Engine.

The following description is based on a local Apache Tomcat installation.

4. Create your first RESTful Webservice

4.1. Create a new Gradle project and configure jersey usage and Eclipse WTP

Create a new Gradle project named *com.vogella.jersey.first* with *com.vogella.jersey.first* as the top-level package name and configure Eclipse WTP. You can follow Required setup for Gradle and Eclipse web projects to get started.

To import the Jersey dependencies, add the following dependency to your build.gradle file.

```
compile 'org.glassfish.jersey.containers:jersey-container-servlet:2.25.1'
```

4.2. Java Class

Create the following class.

```
package com.vogella.jersey.first;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

// Plain old Java Object it does not extend as class or implements
// an interface

// 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 the *text* and the *HTML* MIME types. It also defines via the `@Path` annotation that its service is available under the `hello` URL.

The browser will always request the HTML MIME type. To see the text version, you can use tool like curl.

4.3. Define Jersey Servlet dispatcher

You need to register Jersey as the servlet dispatcher for REST requests.

Open the file `web.xml` and modify it 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"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
  <display-name>com.vogella.jersey.first</display-name>
 <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-
class>
     <!-- Register resources and providers under com.vogella.jersey.first
package. -->
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.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>
```

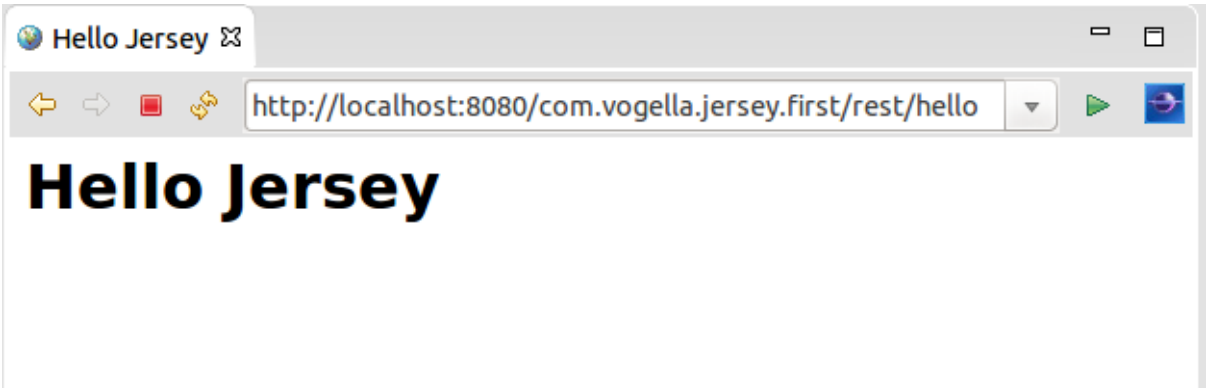
The parameter `jersey.config.server.provider.packages` defines in which package Jersey will look for the web service classes. This property must point to your resources classes. The URL pattern defines the part of the base URL your application will be placed.

4.4. Run your rest service

To run your web application in Eclipse, make sure to run the gradle task `eclipseWtp` first. Afterwards, you should be able to run the application by **right click on the projects name Run As Run on Server**

You should be able to access your resources under the following URL:

```
http://localhost:8080/com.vogella.jersey.first/rest/hello
```



This URL is derived from the *context root* value in the projects properties *Web Project Settings* (by default, this is your application name), augmented with the servlet-mapping URL-pattern and the `hello` `@Path` annotation from your class file. You should get the message "Hello Jersey".

As we are using Gradle, if you want to update the context root include the following in your build.gradle and update your web container (**right click on server Publish** in the *Servers* Eclipse View).

```
eclipse {
    wtp {
        component {
            contextPath = 'newName'
        }
    }
}
```

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.

5. Create a REST client

Jersey contains a REST client library which can be used for testing or to build a real client in Java. The usage of this library is demonstrated in the following tutorial.

Create a new Java gradle project with *com.vogella.jersey.first.client* as top-level package name and add following dependency to your build.gradle file to import the Jersey dependencies.

```
compile 'org.glassfish.jersey.containers:jersey-container-servlet:2.25.1'
```

Create the following test class.

```
package com.vogella.jersey.first.client;

import java.net.URI;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

public class Test {

    public static void main(String[] args) {
        ClientConfig config = new ClientConfig();

        Client client = ClientBuilder.newClient(config);

        WebTarget target = client.target(getBaseURI());

        String response = target.path("rest").
                          path("hello").
                          request().
                          accept(MediaType.TEXT_PLAIN).
                          get(Response.class)
                          .toString();


        String plainAnswer =

target.path("rest").path("hello").request().accept(MediaType.TEXT_PLAIN).ge
t(String.class);
        String xmlAnswer =

target.path("rest").path("hello").request().accept(MediaType.TEXT_XML).get(
String.class);
        String htmlAnswer=
```

```
target.path("rest").path("hello").request().accept(MediaType.TEXT_HTML).get
(String.class);

        System.out.println(response);
        System.out.println(plainAnswer);
        System.out.println(xmlAnswer);
        System.out.println(htmlAnswer);
    }

    private static URI getBaseURI() {
        return
UriBuilder.fromUri("http://localhost:8080/com.vogella.jersey.first").build(
);
    }
}
```

6. RESTful web services 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. For an introduction into JAXB please see JAXB. You can continue this tutorial without reading these tutorials, but they contain more background information.

6.1. Create new Gradle project

Create a new Gradle project named *com.vogella.jersey.jaxb* with *com.vogella.jersey.jaxb* as the top-level package name and configure Eclipse WTP. You can follow Required setup for Gradle and Eclipse web projects to get started. To enable JSON support, add the following dependency to your *build.gradle* file. The second line automatically adds support for the media type `application/json`.

```
compile 'org.glassfish.jersey.containers:jersey-container-servlet:2.25.1'
compile 'org.glassfish.jersey.media:jersey-media-json-jackson:2.25.1'
```

6.2. Create Java classes

Create your domain class.

```
package com.vogella.jersey.jaxb;

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 returns an instance of the `Todo` class.

```
package com.vogella.jersey.jaxb;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/todo")
public class TodoResource {

    // This method is called if XML is requested
    @GET
    @Produces({MediaType.APPLICATION_XML})
    public Todo getXML() {
        Todo todo = new Todo();
        todo.setSummary("Application XML Todo Summary");
        todo.setDescription("Application XML Todo Description");
        return todo;
    }

    // This method is called if JSON is requested
    @GET
    @Produces({MediaType.APPLICATION_JSON})
    public Todo getJSON() {
        Todo todo = new Todo();
        todo.setSummary("Application JSON Todo Summary");
        todo.setDescription("Application JSON Todo Description");
        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("XML Todo Summary");
        todo.setDescription("XML Todo Description");
        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"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
  <display-name>com.vogella.jersey.jaxb</display-name>
 <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-
class>
     <!-- Register resources and providers under com.vogella.jersey.first
package. -->
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.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 the following URL.

```
http://localhost:8080/com.vogella.jersey.jaxb/rest/todo
```

6.3. Create a client

Create a new Java Gradle project with *com.vogella.jersey.jaxbclient* as top-level package name and add the following dependencies to your build.gradle file to import the Jersey dependencies and enable JSON support.

```
compile 'org.glassfish.jersey.containers:jersey-container-servlet:2.25.1'
compile 'org.glassfish.jersey.media:jersey-media-json-jackson:2.25.1'
```

Create the following test class.

```
package com.vogella.jersey.jaxbclient;

import java.net.URI;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

import com.fasterxml.jackson.jaxrs.annotation.JacksonFeatures;

public class TodoTest {
```

```
    public static void main(String[] args) {
        ClientConfig config = new ClientConfig();
        Client client = ClientBuilder.newClient(config);

        WebTarget target = client.target(getBaseURI());
        // Get XML
        String xmlResponse = target.path("rest").path("todo").request()
                .accept(MediaType.TEXT_XML).get(String.class);
        // Get XML for application
        String xmlAppResponse =target.path("rest").path("todo").request()
                .accept(MediaType.APPLICATION_XML).get(String.class);

        // Get JSON for application
        String jsonResponse = target.path("rest").path("todo").request()
                .accept(MediaType.APPLICATION_JSON).get(String.class);

        System.out.println(xmlResponse);
        System.out.println(xmlAppResponse);
        System.out.println(jsonResponse);
    }

    private static URI getBaseURI() {
        return UriBuilder.fromUri(
                "http://localhost:8080/com.vogella.jersey.jaxb").build();
    }

}
```

7. 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.

7.1. Project

Create a new Gradle project called *com.vogella.jersey.todo* with *com.vogella.jersey.todo* as top-level package name. Add the following to dependency to your build.gradle file and make also sure, that you enabled Eclipse WTP support by following Required setup for Gradle and Eclipse web projects.

```
compile 'org.glassfish.jersey.containers:jersey-container-servlet:2.25.1'
compile 'javax.servlet:javax.servlet-api:4.0.0-b07'
```

Change the `web.xml` 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"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
 id="WebApp_ID" version="3.0">
  <display-name>com.vogella.jersey.todo</display-name>
 <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
```

```
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-
class>
      <!-- Register resources and providers under com.vogella.jersey.first
package. -->
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.vogella.jersey.todo</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 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 to learn about JAXB.

```
package com.vogella.jersey.todo;


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 com.vogella.jersey.todo;

import java.util.HashMap;
import java.util.Map;

import com.vogella.jersey.todo.Todo;

public enum TodoDao {
    instance;

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

    private TodoDao() {

        Todo todo = new Todo("1", "Learn REST");
        todo.setDescription("Read
https://www.vogella.com/tutorials/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;
    }

}
```

7.2. 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 called `create_todo.html` in the `WebContent` folder.

```
<!DOCTYPE html>
<html>
 <head>
  <title>Form to create a new resource</title>
 </head>
<body>
    <form action="../com.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>
```

7.3. Rest Service

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

```
package com.vogella.jersey.todo;

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 com.vogella.jersey.todo.TodoDao;
import com.vogella.jersey.todo.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 com.vogella.jersey.todo;

import java.io.IOException;
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.UriInfo;

import com.vogella.jersey.todo.TodoDao;
import com.vogella.jersey.todo.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/com.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/com.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 the `@PathParam` annotation to define that the `id` is inserted as parameter.

7.4. Run

Run your web application in Eclipse and test the availability of your REST service under:

```
http://localhost:8080/com.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/com.vogella.jersey.todo/rest/todos/count
```

to see an existing TODO use

```
http://localhost:8080/com.vogella.jersey.todo/rest/todos/{id}"
, e.g.,
http://localhost:8080/com.vogella.jersey.todo/rest/todos/1
```

to see the TODO with ID `1`. We currently have only TODOs with the ids 1 and 2, all other requests will result in 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.

7.5. Create a client

To test your service you can create a new class in your server project. This project has already all required libs in the classpath, so this is faster than creating a new project.

Create the following class.

```
package com.vogella.jersey.todo;

import java.net.URI;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.Form;
import javax.ws.rs.core.MediaType;
```

```
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

import com.vogella.jersey.todo.Todo;

public class Tester {
  public static void main(String[] args) {


      ClientConfig config = new ClientConfig();
      Client client = ClientBuilder.newClient(config);
      WebTarget service = client.target(getBaseURI());

      // create one todo
      Todo todo = new Todo("3", "Blabla");
      Response response =
service.path("rest").path("todos").path(todo.getId()).request(MediaType.APP
LICATION_XML).put(Entity.entity(todo,MediaType.APPLICATION_XML),Response.cl
ass);

      // Return code should be 201 == created resource
      System.out.println(response.getStatus());

      // Get the Todos

System.out.println(service.path("rest").path("todos").request().accept(Medi
aType.TEXT_XML).get(String.class));

//    // Get JSON for application (Make sure to add the jersey-media-json-
jackson dependency to add support for JSON)
//
System.out.println(service.path("rest").path("todos").request().accept(Medi
aType.APPLICATION_JSON).get(String.class));

      // Get XML for application

System.out.println(service.path("rest").path("todos").request().accept(Medi
aType.APPLICATION_XML).get(String.class));

      //Get Todo with id 1
      Response checkDelete =
service.path("rest").path("todos/1").request().accept(MediaType.APPLICATION
_XML).get();

      //Delete Todo with id 1
      service.path("rest").path("todos/1").request().delete();

      //Get get all Todos id 1 should be deleted

System.out.println(service.path("rest").path("todos").request().accept(Medi
aType.APPLICATION_XML).get(String.class));

      //Create a Todo
      Form form =new Form();
      form.param("id", "4");
      form.param("summary","Demonstration of the client lib for forms");
      response =
service.path("rest").path("todos").request().post(Entity.entity(form,MediaT
ype.APPLICATION_FORM_URLENCODED),Response.class);
```

```
      System.out.println("Form response " + response.getStatus());

      //Get all the todos, id 4 should have been created

System.out.println(service.path("rest").path("todos").request().accept(Medi
aType.APPLICATION_XML).get(String.class));

  }

  private static URI getBaseURI() {
    return
UriBuilder.fromUri("http://localhost:8080/com.vogella.jersey.todo").build()
;
  }
}
```