Running the Advanced JAX-RS Example Application

This chapter describes how to build and run the `customer` sample application.

This example application is a RESTful web service that uses JAXB to perform the Create, Read, Update, Delete (CRUD) operations for a specific entity.

The `customer` Example Application

The `customer` sample application is in the *tut-install*`/examples/jaxrs/customer` directory.

See Chapter 2, Using the Tutorial Examples for basic information on building and running sample applications.

The `customer` Application Source Files

The source files of this application are at *tut-install*/`examples/jaxrs/customer/src/java`.

This application has the following source files:

- The `CustomerService` Class
- The XSD Schema for the `customer` Application

The CustomerService Class

```
@Path("/Customer")
public class CustomerService{
public static final String
DATA_STORE = "CustomerDATA.txt";
public static final Logger logger =
Logger.getLogger(CustomerService.
class.getCanonicalName());
@POST
@Consumes("application/xml")
```

```
public Response createCustomer
(CustomerType customer){
try {
long customerId =
persist(customer);
return Response.created(URI.
create("/" + customerId)).build();
} catch (Exception e) {
throw new
WebApplicationException(e,
Response.Status.INTERNAL_SERVER_ERROR);
}
}
private long persist
(CustomerType customer)
throws IOException {
File dataFile =
new File(DATA_STORE);

if (!dataFile.exists())
{dataFile.createNewFile();}
long customerId = customer.getId();
```

```
Properties properties =
new Properties();
properties.load
(new FileInputStream(dataFile));
properties.setProperty
(String.valueOf(customerId),
customer.getFirstName() + "," +
customer.getLastName()
+ "," + customer.getEmail() + ","
+ customer.getCity()
+ "," + customer.getCountry());
```

```
properties.store
 (new FileOutputStream
 (DATA_STORE),null);
return customerId;
 }
 }
```

The `CustomerService` class has a `createCustomer` method that creates a customer resource based on the `CustomerType` and returns a URI for the new resource.

The `persist` method emulates the behavior of the JPA entity manager.

This example uses a `java.util.Properties` file to store data.

If you are using the default configuration of GlassFish, the properties file is at *as-install*`/glassfish/domains/domain1/CustomerDATA.txt`.

The response that is returned to the client has a URI to the newly created resource.

The return type is an entity body mapped from the property of the response with the status code specified by the status property of the response.

The `WebApplicationException` is a `RuntimeException` that is used to wrap the appropriate HTTP error status code, such as 404, 406, 415, or 500.

The `@Consumes("application/xml")` and `@Produces("application/xml")` annotations set the request and response media types to use the appropriate MIME client.

These annotations can be applied to a resource method, a resource class, or even to an entity provider.

If you do not use these annotations, JAX-RS allows the use of any media type ("*/*").

The following code snippet shows the implementation of the `getCustomer` and `findbyId` methods.

The `getCustomer` method uses the `@Produces`annotation and returns a `JAXBElement` with a `CustomerType` object, which is a JAXB XML based entity, generated through the `xjc` binding compiler.

```
@GET
@Path("{id}")
@Produces("application/xml")
public JAXBElement<CustomerType>
getCustomer(
@PathParam("id")String customerId){
CustomerType customer = null;
try
{customer = findById(customerId);}
catch (Exception ex) {
```

```
logger.log(Level.SEVERE,
"Error calling searchCustomer()
for customerId {0}.{1}",
new Object[]{customerId,
ex.getMessage()});
}
return
new ObjectFactory().createCustomer
(customer);
}
```

```
private CustomerType findById
(String customerId)
throws IOException {
properties properties =
new Properties();
properties.load(
new FileInputStream(DATA_STORE));
String rawData =
properties.getProperty(customerId);
if (rawData != null) {
```

```
final String[] field =
rawData.split(",");
ObjectFactory objFactory =
new ObjectFactory();
CustomerType customer =
objFactory.createCustomerType();
customer.setFirstName(field[0]);
customer.setLastName(field[1]);
customer.setEmail(field[2]);
customer.setCity(field[3]);
customer.setCountry(field[4]);
customer.setId
(Integer.parseInt(customerId));
return customer;
}
return null;
}
```

The XSD Schema for the `customer` Application

A sample XSD schema for the `Customer` entity is as follows:

```
<?xml version="1.0"?>
<xs:schema
targetNamespace=
"http://examples.oracle.com"
xmlns:xs=
"http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
xmlns:ora=
"http://examples.oracle.com"
>
```

```
<xs:element name="customer"
type="ora:CustomerType"
/>
<xs:complexType name="CustomerType">
<xs:sequence>
<xs:element name="id"
type="xs:int"/>
<xs:element name="firstName"
type="xs:string"/>
<xs:element name="lastName"
type="xs:string"/>
```

```
<xs:element name="city"
type="xs:string"/>
<xs:element name="country"
type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

The `CustomerClient` Class

Jersey is the reference implementation of JAX-RS (JSR 311).

You can use the Jersey client API to write a test client for the `customer` example application.

You can find the Jersey APIs at http://jersey.java.net/nonav/apidocs/latest/jersey/.

The CustomerClient class calls Jersey APIs to test the CustomerService web service:

```
package customer.rest.client;
import
com.oracle.examples.CustomerType;
```

```
import
com.oracle.examples.ObjectFactory;
import
com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.
ClientResponse;
import com.sun.jersey.api.client.
WebResource;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBElement;
public class CustomerClient{
public static final Logger logger =
Logger.getLogger(CustomerClient.
class.getCanonicalName());
public static void main
(String[] args){
Client client = Client.create();
// Define the URL for testing
// the example application
```

```
WebResource webResource =
client.resource("http://
localhost:8080/customer/Customer");
ObjectFactory factory =
new ObjectFactory();
// Test the POST method
CustomerType customerType =
new CustomerType();
customerType.setId(1);
customerType.setFirstName("Duke");
customerType.setLastName("OfJava");
customerType.setCity("JavaTown");
customerType.setCountry("USA");
JAXBElement<CustomerType> customer
=
factory.createCustomer(customerType);
ClientResponse response =
webResource.type("application/xml")
.post(ClientResponse.class,customer);
logger.info("POST status: {0}" +
response.getStatus());
if (response.getStatus() == 201){
```

```
logger.info("POST succeeded");
} else
{logger.info("POST failed");}
// Test the GET method using
// content negotiation
response = webResource.path("1").
accept(MediaType.APPLICATION_XML)
.get(ClientResponse.class);
CustomerType entity =
response.getEntity
(CustomerType.class);
```

```
logger.info("GET status: " +
response.getStatus());
if (response.getStatus() == 200){
logger.info(
"GET succeeded, city is " +
entity.getCity());
} else
{ logger.info("GET failed"); }
// Test the DELETE method
response = webResource.path("1").
delete(ClientResponse.class);
```

```
logger.info("DELETE status: " +
response.getStatus());
if (response.getStatus() == 204){
logger.info
("DELETE succeeded (no content)");
} else
{ logger.info("DELETE failed"); }
response = webResource.path("1").
accept(MediaType.APPLICATION_XML)
.get(ClientResponse.class);
```

```
entity = response.getEntity
(CustomerType.class);
logger.info("GET status: " +
response.getStatus());
try
{logger.info(entity.getCity());}
catch (NullPointerException ne){
// as expected, returns null
// because you have deleted
// the customer
logger.info
("After DELETE, city is: " +
ne.getCause());
}
}
}
```

This Jersey client tests the POST, GET, and DELETE methods.

All of these HTTP status codes indicate success: 201 for POST, 200 for GET, and 204 for DELETE.

For details about the meanings of HTTP status codes, see http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

Building, Packaging, Deploying, and Running the `customer` Example

You can build, package, deploy, and run the `customer` application by using either NetBeans IDE or the Ant tool.

To Build, Package, and Deploy

the `customer` *Example Using NetBeans IDE*

This procedure builds the application into the following directory:

tut-install`/examples/jax-rs/customer/build/web`

The contents of this directory are deployed to the GlassFish Server.

1. From the File menu, choose Open Project.

2. In the Open Project dialog, navigate to:
`tut-install``/examples/jaxrs/`

3. Select the `customer` folder.

4. Select the Open as Main Project check box.

5. Click Open Project.

It may appear that there are errors in the source files, because the files refer to JAXB classes that will be generated when you build the application.

You can ignore these errors.

6. In the Projects tab, right-click the `customer` project and select Deploy.

To Build, Package, and Deploy the `customer` Example Using Ant

1. In a terminal window, go to:

```
tut-install/examples/jaxrs/
customer/
```

2. Type the following command:

```
ant
```

This command calls the `default` target, which builds and packages the application into a WAR file, `customer.war`, located in the `dist` directory.

3. Type the following command:

```
ant deploy
```

Typing this command deploys `customer.war` to the GlassFish Server.

To Run the `customer` Example Using the Jersey Client

1. In NetBeans IDE, expand the Source Packages node.

2. Expand the `customer.rest.client` node.

3. Right-click the `CustomerClient.java` file and click Run File.

The output of the client looks like this:

```
run:
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
INFO: POST status: 201
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
INFO: POST succeeded
```

Running the Advanced JAX-RS Example Application

```
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
INFO: GET status: 200
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
INFO: GET succeeded, city is JavaTown
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
INFO: DELETE status: 204
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
INFO: DELETE succeeded (no content)
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
INFO: GET status: 200
Jan 18, 2011 2:40:20 PM
customer.rest.client.CustomerClient main
```

Running the Advanced JAX-RS Example Application

```
INFO: After DELETE, city is: null
BUILD SUCCESSFUL (total time: 5 seconds)
```

To Run the `customer` Example Using the Web Services Tester

1. In NetBeans IDE, right-click the customer node and click Test RESTful Web Services.

The step deploys the application and brings up a test client in the browser.

2. When the test client appears, select the Customer resource node in the left pane.

3. Paste the following XML code into the Content text box, replacing “Insert content here”:

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

```
<ora:customer
xmlns:ora=
"http://examples.oracle.com"
xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://examples.oracle.com Customer.xsd">
<ora:id>1</ora:id>
<ora:firstName>Duke</ora:firstName>
<ora:lastName>OfJava</ora:lastName>
<ora:city>JavaTown</ora:city>
<ora:country>USA</ora:country>
```

```
</ora:customer>
```

You can find the code in the file `customer/sample-input.txt`.

4. Click Test.

The following message appears in the window below:

`Status: 201 (Created)`

Below it is a `POST RequestFailed` message, which you can ignore.

5. Expand the Customer node and click `{id}`.
6. Type `1` in the id field and click Test to test the GET method.

The following status message appears:

```
Status: 200 (OK)
```

The XML output for the resource appears in the Response window:

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

```
<customer
xmlns="http://examples.oracle.com"
>
<id>1</id>
<firstName>Duke</firstName>
<lastName>OfJava</lastName>
<city>JavaTown</city>
<country>USA</country>
</customer>
```

A GET for a nonexistent ID also returns a `200 (OK)` status, but the output in the Response window shows no content:

```
<?xml version="1.0"
encoding="UTF-8"?>
<customer
xmlns="http://examples.oracle.com"
xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:nil="true"/>
```

You can test other methods as follows:

7. Click PUT, type the input for an existing customer, modify any content except the `id` value, and click Test to update the customer fields.

A successful update returns the following status message:

```
Status: 303 (See Other)
```

- Click DELETE, type the ID for an existing customer, and click Test to remove the customer.

A successful delete returns the following status message:

```
Status: 303 (See Other)
```

Using Curl to Run the `customer` Example Application

Curl is a command-line tool that you can use to run the customer application on UNIX platforms.

You can download Curl from http://curl.haxx.se or add it to a Cygwin installation.

To add a new customer and test the POST method, use the following command:

```
curl -i --data @sample-input.txt \
--header Content-type:application/xml
http://localhost:8080/customer/Customer
```

A successful POST returns an HTTP `Status:` `201 (Created)` status.

To retrieve the details of the customer whose id is 1, use the following command:

```
curl -i -X GET
http://localhost:8080/customer/Customer/1
```

A successful GET returns an HTTP `Status`: `200 (OK)` status.

To delete a customer record, use the following command:

```
curl -i -X DELETE
http://localhost:8080/customer/Customer/1
```

A successful DELETE returns an HTTP `Status: 303` status.