12

Connecting to a RESTful Web Service

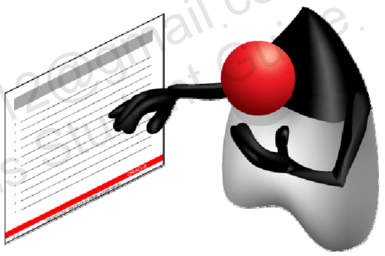
Objectives

After completing this lesson, you should be able to:

- Describe how to test a RESTful Web Service
- Identify how to develop a Jersey RESTful Client
- Review the implementation of web service clients in the HenleyApp application

Topics

- Test RESTful web services
- Steps to develop JAX-RS web service clients
- Web service clients of HenleyApp



Testing a RESTful Web Service

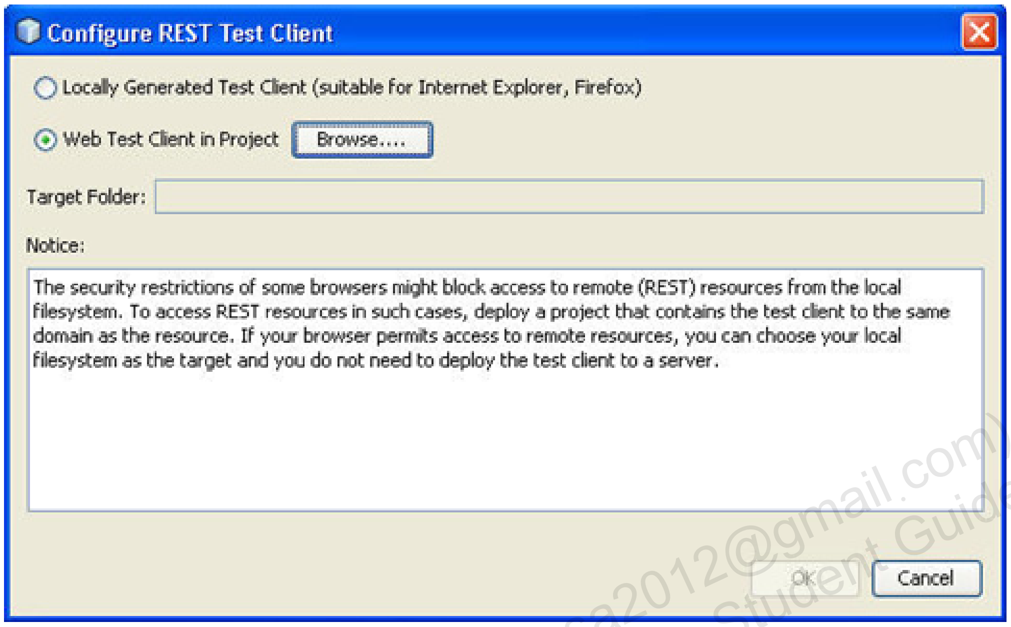
Check the following while testing a RESTful web service:

- That the URL address correctly represents the service deployment endpoint and the method annotations
- That the server requests(GET, PUT, DELETE, or POST) that are generated invoke the appropriate methods of the web service.
- That the methods return acceptable data

There are various tools and software to test a RESTful web service. NetBeans IDE also provides various options to test a RESTful web service within the same project.

Later in this lesson, a slide shows the service requests that are generated for each resource.

Testing HenleyServer's RESTful Web Services


Configure REST Test Client
Locally Generated Test Client (suitable for Internet Explorer, Firefox)
Web Test Client in Project
Browse....
Target Folder:
Notice:
OK
Cancel

Locally Generated Test Client (suitable for Internet Explorer, Firefox) [✓ Web Test Client in Project]

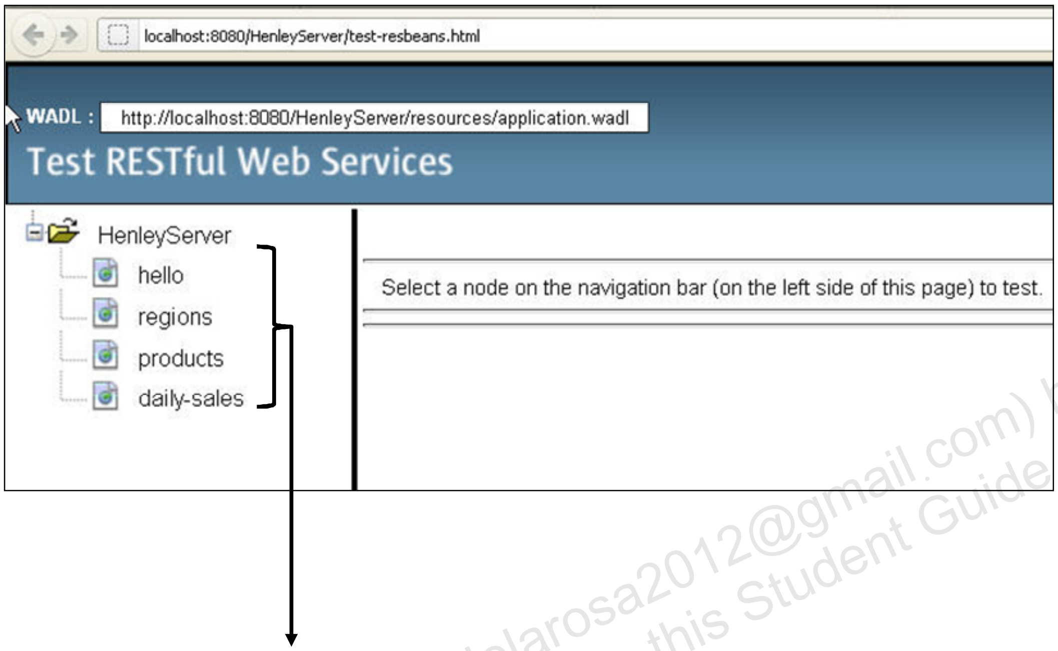
The security restrictions of some browsers might block access to remote (REST) resources from the local filesystem. To access REST resources in such cases, deploy a project that contains the test client to the same domain as the resource. If your browser permits access to remote resources, you can choose your local filesystem as the target and you do not need to deploy the test client to a server.

To test a RESTful web service in NetBeans:

- Right-click the project node and select Test RESTful Web Services. A dialog box opens asking whether you want to generate the test client inside the service project or in another Java web project. This option lets you work around security restrictions in some browsers. You can use any web project, as long as it is configured to deploy in the same server domain as the HenleyServer project.

The screenshot in the slide shows the two options for the target location of the generated test client.

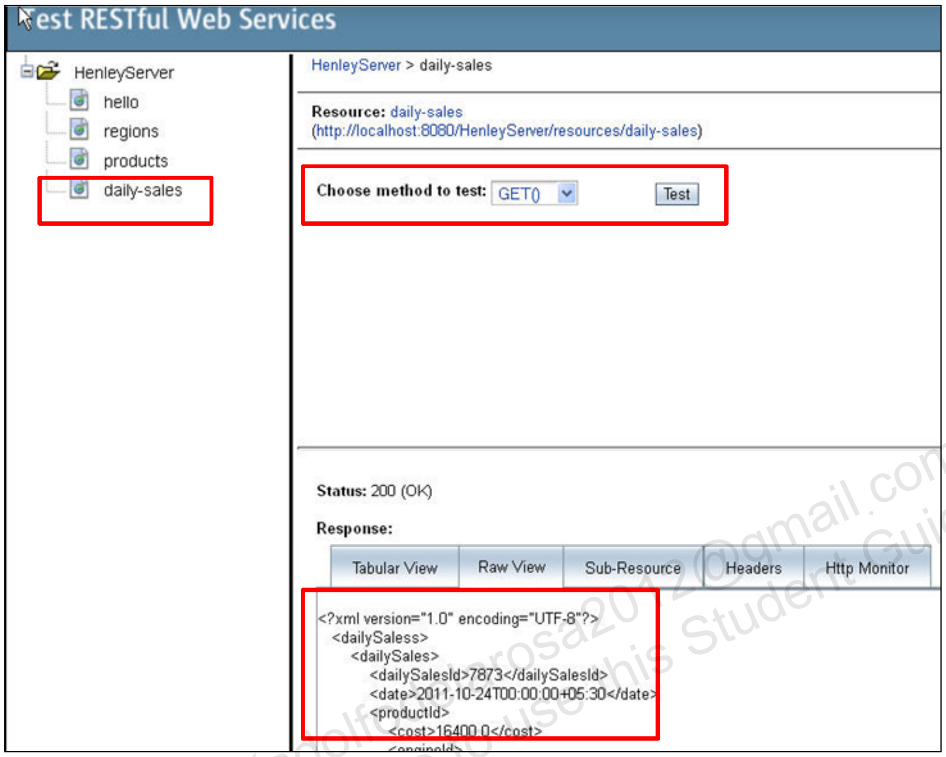
Testing HenleyServer’s RESTful Web Services


localhost:8080/HenleyServer/test-resbeans.html
WADL : http://localhost:8080/HenleyServer/resources/application.wadl
Test RESTful Web Services
HenleyServer
hello
regions
products
daily-sales
Select a node on the navigation bar (on the left side of this page) to test.

On the left side is the set of root resources.

After you have selected where to generate the test client, click OK. The server starts and the application is deployed. When deployment is complete, the browser displays your application, with a link for each of the web services.

Testing HenleyServer's RESTful Web Services



Resource: daily-sales (http://localhost:8080/HenleyServer/resources/daily-sales)

Status: 200 (OK)

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <dailySaless>
    <dailySales>
      <dailySalesId>7873</dailySalesId>
      <date>2011-10-24T00:00:00+05:30</date>
      <productId>
        <cost>16400.0</cost>
```

Click one resource node. In the "Choose method to test" field, select either GET or POST, and then click Test. The test client sends a request and displays the result in the Test Output section.

The test client displays the Raw View by default. The image in the slide shows the response to an application/JSON request.

Testing HenleyServer's RESTful Web Services

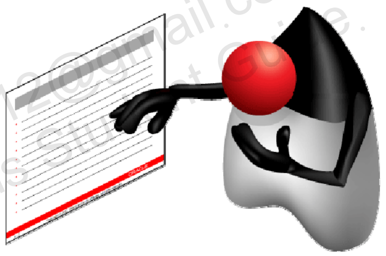
Tab Name	Description
Tabular View	A flattened view that displays all the URIs in the resulting document. Currently, this view only displays a warning that Container-Containee relationships are not allowed.
Raw View	Displays the actual data returned. Depending on which mime type you selected (application/xml), the data displayed will be XML format, respectively.
Sub-Resource	Shows the URLs of the root resource and sub-resources. When the RESTful web service is based on database entity classes, the root resource represents the database table, and the sub-resources represent the columns.
Headers	Displays the HTTP header information
Http Monitor	Displays the actual HTTP requests and responses sent and received

There are five tabs in the Test Output section of the Test Client window, as shown in the table in the slide.

Exit the browser and return to the IDE.

Topics

- Test RESTful web services
- Steps to develop JAX-RS web service clients
- Web service clients of HenleyApp

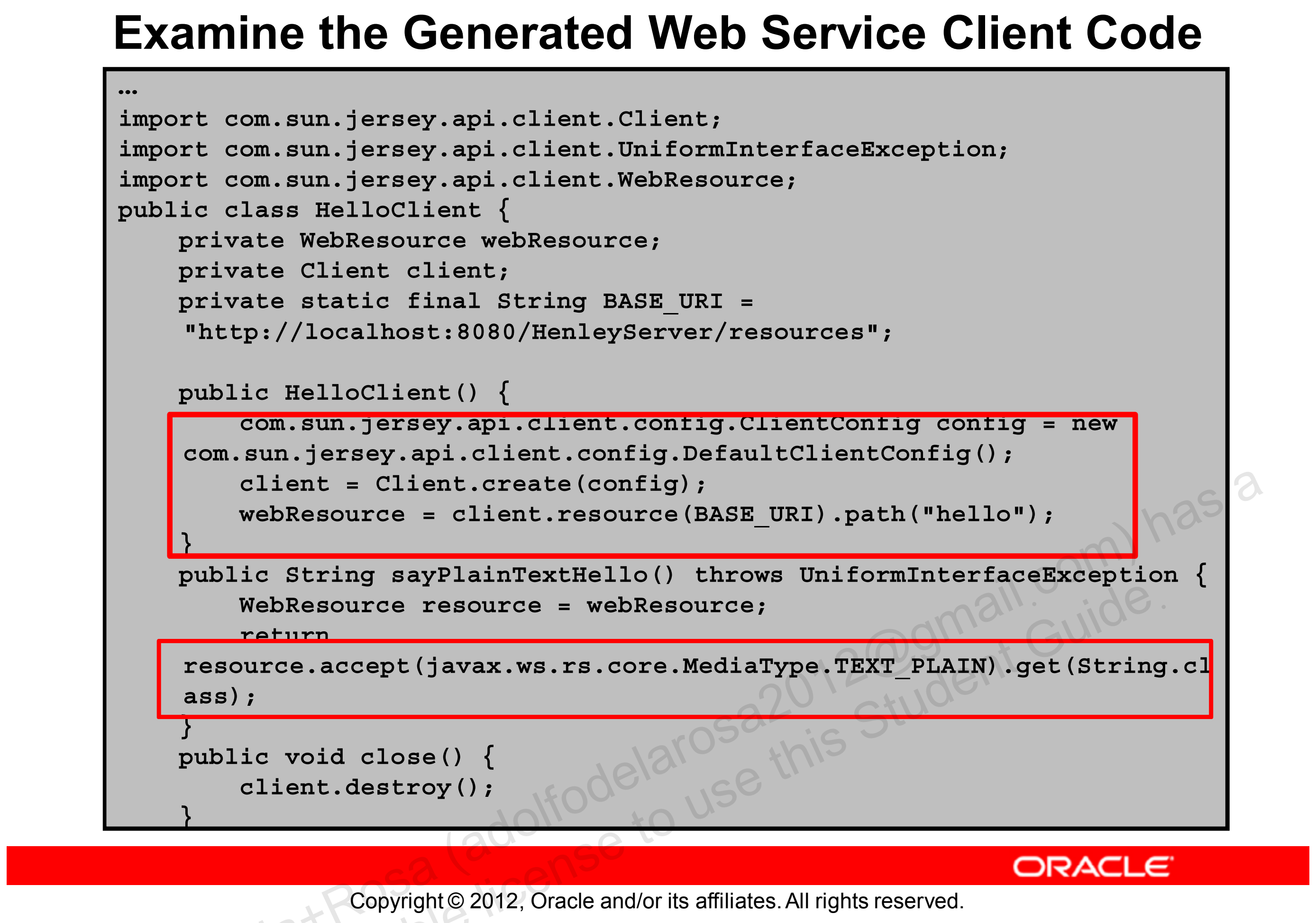


Steps to Develop a Restful Web Service Client

1. Ensure that the project has the required libraries added:
   - JAX-RI
   - Jersey
2. Identify the GUI window and control where the results of the web service invocation will be displayed.
3. Create a new RESTful service client using a wizard. The following information is important:
   - The URL of the RESTful service
   - The package name
   - The class where the client code will be generated
4. Invoke the generated code in the GUI window appropriately.

The slide lists the steps to develop a web service client. You can write code to consume a RESTful web service in any Java SE/Java web or in a JavaFX application. NetBeans provides a wizard to easily do the same.

You must know the URL of the service that you want to consume in your client program.



Examine the Generated Web Service Client Code

```
…
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.UniformInterfaceException;
import com.sun.jersey.api.client.WebResource;
public class HelloClient {
    private WebResource webResource;
    private Client client;
    private static final String BASE_URI =
    "http://localhost:8080/HenleyServer/resources";

    public HelloClient() {
        com.sun.jersey.api.client.config.ClientConfig config = new
    com.sun.jersey.api.client.config.DefaultClientConfig();
        client = Client.create(config);
        webResource = client.resource(BASE_URI).path("hello");
    }
    public String sayPlainTextHello() throws UniformInterfaceException {
        WebResource resource = webResource;
        return
    resource.accept(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.cl
    ass);
    }
    public void close() {
        client.destroy();
    }
```

The code snippet in the slide shows NetBeans-generated code.

Topics

- Test RESTful web services
- Steps to develop JAX-RS web service clients
- Web service clients of HenleyApp

Apply the Web Service Client Code in HenleyClient

```
public class GetDailySalesTask extends Task<ObservableList<DailySales>> {
    private static final ClientConfig CONFIG = new DefaultClientConfig();
    static {
        CONFIG.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING,
Boolean.TRUE);
    }
    @Override protected ObservableList<DailySales> call() throws Exception
{
        Client client = Client.create(CONFIG);
        WebResource webResource =
client.resource("http://localhost:8080/HenleyServer/resources").path("dail
y-sales");
   ClientResponse response = webResource.get(ClientResponse.class);
        GenericType<List<DailySales>> genericType = new
GenericType<List<DailySales>>() {};
        // Return an ObservableList backed by the ArrayList of DailySales
from the web service
        return
FXCollections.observableArrayList(response.getEntity(genericType));
    }
}
```

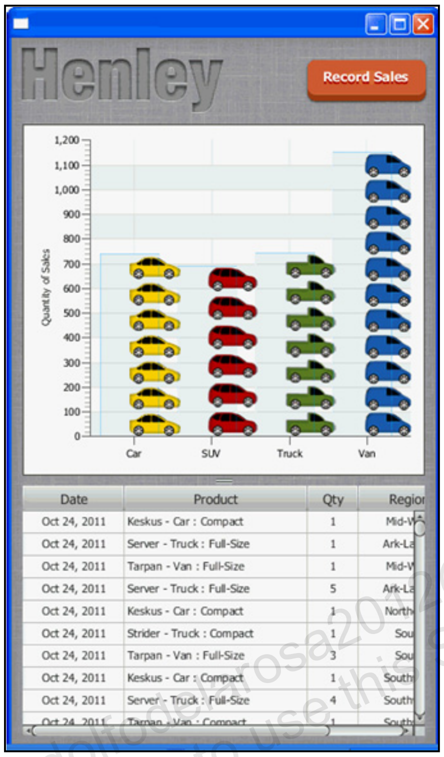
The code to access the web service URL and invoke the GET operation

ORACLE

The lines of code in the slide is from `GetDailySalestask.java` of the `HenleyClient4` project.

The `webResource.get(ClientResponse.class)` invokes the `findAll` method of the daily-sales web service.

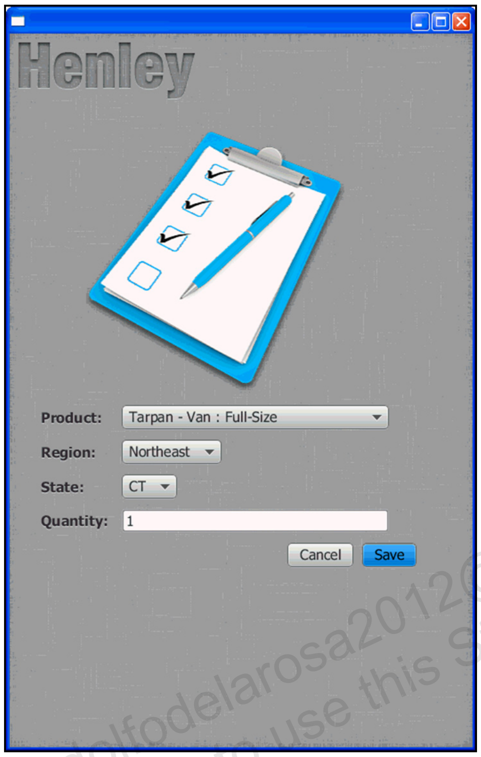
Verify the Output in HenleyClient



Date	Product	Qty	Regior
Oct 24, 2011	Keskus - Car : Compact	1	Mid-W
Oct 24, 2011	Server - Truck : Full-Size	1	Ark-La
Oct 24, 2011	Tarpan - Van : Full-Size	1	Mid-W
Oct 24, 2011	Server - Truck : Full-Size	5	Ark-La
Oct 24, 2011	Keskus - Car : Compact	1	North
Oct 24, 2011	Strider - Truck : Compact	1	Sou
Oct 24, 2011	Tarpan - Van : Full-Size	3	Sou
Oct 24, 2011	Keskus - Car : Compact	1	South
Oct 24, 2011	Server - Truck : Full-Size	4	South
Oct 24, 2011	Tarpan - Van : Compact	1	South

ORACLE

Sales Form to Insert Details



Product:	Tarpan - Van : Full-Size
Region:	Northeast
State:	CT
Quantity:	1

The user provides the sales information in the sales form (`SalesForm.fxml`) as shown in the slide, and then clicks the Save button. The `SalesForm.java` class is executed.

Examining the Code for the POST/Insert Operation

The daily-sales web service URL

```
 …
final Task<List<Product>> saveSaleTask = new Task() {
       @Override protected Object call() throws Exception
{
       Client client = Client.create(CONFIG);
       WebResource dailySalesWebResource =
client.resource("http://localhost:8080/HenleyServer/resou
rces").path("daily-sales");

dailySalesWebResource.type(javax.ws.rs.core.MediaType.APP
LICATION_JSON).post(sale);
                return null;
            }
```

The POST operation/create method/insert statement (for a dialy_sale record) is invoked.

The code snippet in the slide is from SalesForm.java of HenleyClient4 project:

1. A task is started, which invokes the call method.
2. The `daily-sales` web service is consumed through its URL.
3. The `create` or the `post` method of the `daily-sales` service is invoked.
4. The `create` method inserts the from data as a record in the DailySales table.

These steps execute when the user clicks the Save button of the `SalesForm.fxml` form.

By invoking the `Dashboard.fxml` file, you can verify the record insertion.

Quiz

What do you need to create a RESTful client? Select from the list below:

a. The URL of the RESTful service
b. Names of all the methods that are exposed by the web service
c. JAX-RS RI and Jersey libraries
d. A HTML web page to invoke the web service

Answer: a, c

Summary

In this lesson, you should have learned how to:

- Describe how to test a RESTful Web Service
- Identify how to develop a Jersey RESTful Client
- Review the implementation of web service clients in the HenleyApp application

Practice 12: Overview

- Practice 12-1: Testing RESTful Web Services
- Practice 12-2: Creating a RESTful Web Service Client