4 Developing RESTful Web Service Clients

This chapter describes how to develop WebLogic Web service clients that conform to the Representational State Transfer (REST) architectural style using Java API for RESTful Web Services (JAX-RS).

This chapter includes the following sections:

- About RESTful Web Service Client Development
- Creating and Configuring a Client Instance
- Creating a Web Resource Instance
- Sending Requests to the Resource
- Receiving a Response from a Resource

About RESTful Web Service Client Development

The Jersey JAX-RS RI provides a client API for developing RESTful Web services clients. To access the client API, you create an instance of the `com.sun.jersey.api.client.Client` class and then use that instance to access the Web resource and send HTTP requests.

> **Note:**
> A standard client API will be supported as part of the JSR-311 JAX-RS 2.0 specification.

The following sections provide more information about RESTful Web service client development:

- Summary of Tasks to Develop RESTful Web Service Clients
- Example of a RESTful Web Service Client

Summary of Tasks to Develop RESTful Web Service Clients

The following table summarizes a subset of the tasks that are required to develop RESTful Web service clients. For more information about advanced tasks, see More Advanced RESTful Web Service Client Tasks.

Table 4-1 Summary of Tasks to Develop RESTful Web Service Clients

Task	More Information
Create an instance of the `com.sun.jersey.api.client.Client` class.	Creating and Configuring a Client Instance
Create an instance of the Web resource.	Creating a Web Resource Instance
Send requests to the resource. For example, HTTP requests to GET, PUT, POST, and DELETE resource information.	Sending Requests to the Resource
Receive responses from the resource.	Receiving a Response from a Resource

Example of a RESTful Web Service Client

The following provides a simple example of a RESTful Web service client that can be used to call the RESTful Web service defined in Example 2-1, "Simple RESTful Web Service" (develop.htm#BABFEBGI). In this example:

- The `Client` instance is created to access the client API. For more information, see Creating and Configuring a Client Instance.
- The `WebResource` instance is created to access the Web resource. For more information, see Creating a Web Resource Instance.
- A `get` request is sent to the resource. For more information, see Sending Requests to the Resource.
- The response is returned as a String value. For more information about receiving the response, see Receiving a Response from a Resource.

Additional examples are listed in Learn More About RESTful Web Services. (overview.htm#CEGBHFHF)

Example 4-1 Simple RESTful Web Service Client Example

Download


```
package samples.helloworld.client;

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

public class helloWorldClient {
    public helloWorldClient() {
        super();
    }

    public static void main(String[] args) {
        Client c = Client.create();
        WebResource resource = c.resource("http://localhost:7101/RESTfulService-Project1-context-root/jersey/helloWorld");
        String response = resource.get(String.class);
    }
}
```

() ()

Creating and Configuring a Client Instance

To access the Jersey JAX-RS RI client API, create an instance of the **com.sun.jersey.api.client.Client** class. For more information, see **http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/Client.html** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/Client.html

Optionally, you can pass client configuration properties when creating the client instance, as defined in Table 4-2, by defining a **com.sun.jersey.api.client.Client.ClientConfig** and passing the information to the **create** method. For more information, see **http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/ClientConfig.html** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/Clie

() () ***Table 4-2 RESTful Web Service Client Configuration Properties***

Property	Description
PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION	Boolean value that specifies whether the client should buffer the response entity, if any, and close resources when a **UniformInterfaceException** is thrown. This property defaults to **true**.
PROPERTY_CHUNKED_ENCODING_SIZE	Integer value that specifies the chunked encoding size. A value equal to or less than 0 specifies that the default chunk size should be used. If not set, then chunking will not be used.
PROPERTY_CONNECT_TIMEOUT	Integer value that specifies the connect timeout interval in milliseconds. If the property is 0 or not set, then the interval is set to infinity.
PROPERTY_FOLLOW_REDIRECTS	Boolean value that specifies whether the URL will redirect automatically to the URI declared in **3xx** responses. This property defaults to **true**.
PROPERTY_READ_TIMEOUT	Integer value that specifies the read timeout interval in milliseconds. If the property is 0 or not set, then the interval is set to infinity.

Example 4-2 provides an example of how to create a client instance.

() () ***Example 4-2 Creating a Client Instance***

```
import com.sun.jersey.api.client.Client;
...
    public static void main(String[] args) {
        Client c = Client.create();
...
```

Example 4-3 provides an example of how to create a client instance and pass configuration properties to the **create** method.

() () ***Example 4-3 Creating and Configuring a Client Instance***

```
import com.sun.jersey.api.client.*
...
    public static void main(String[] args) {
        ClientConfig cc = new DefaultClientConfig();
        cc.getProperties().put(ClientConfig.PROPERTY_FOLLOW_REDIRECTS, true);
        Client c = Client.create(cc);
...
```

Alternatively, you can configure a client instance after the client has been created, by setting properties on the map returned from the **getProperties** method or calling a specific setter method.

Example 4-4 provides an example of how to configure a client after it has been created. In this example:

- **PROPERTY_FOLLOW_REDIRECTS** is configured by setting the property on the map returned from the **getProperties** method.
- **PROPERTY_CONNECT_TIMEOUT** is configured using the setter method.

() () ***Example 4-4 Configuring a Client Instance After It Has Been Created***

```
import com.sun.jersey.api.client.*
...
    public static void main(String[] args) {
        Client c = Client.create();
        c.getProperties().put(ClientConfig.PROPERTY_FOLLOW_REDIRECTS, true);
        c.setConnectTimeout(3000);
...
```

() ()

Creating a Web Resource Instance

Before you can issue requests to a RESTful Web service, you must create an instance of **com.sun.jersey.api.client.WebResource** or **com.sun.jersey.api.client.AsyncWebResource** to access the resource specified by the URI. The **WebResource** or **AsyncWebResource** instance inherits the configuration defined for the client instance. For more information, see:

- **WebResource**: **http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/WebResource.html** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/Web
- **AsyncWebResource**: **http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/AsyncWebResource.html** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/clie

> **Note:**

Download

> Because clients instances are expensive resources, if you are creating multiple Web resources, it is recommended that you re-use a single client instance whenever possible.

Example 4-5 provides an example of how to create an instance to a Web resource hosted at **`http://example.com/helloworld`**.

*() () **Example 4-5 Creating a Web Resource Instance***

```
import com.sun.jersey.api.client.*
...
    public static void main(String[] args) {\
...
        Client c = Client.create();
        WebResource resource = c.resource("http://example.com/helloWorld");
...
```

Example 4-5 provides an example of how to create an instance to a Web resource hosted at **`http://example.com/helloworld`**.

*() () **Example 4-6 Creating an Asynchronous Web Resource Instance***

```
import com.sun.jersey.api.client.*;
...
    public static void main(String[] args) {\
...
        Client c = Client.create();
         AsyncWebResource asyncResource = c.resource("http://example.com/helloWorld");
...
```

() ()

Sending Requests to the Resource

Use the **`WebResource`** or **`AsyncWebResource`** instance to build requests to the associated Web resource, as described in the following sections:

- How to Build Requests
- How to Send HTTP Requests
- How to Configure the Accept Header
- How to Pass Query Parameters

() ()

How to Build Requests

Requests to a Web resource are structured using the builder pattern, as defined by the **`com.sun.jersey.api.client.RequestBuilder`** interface. The **`RequestBuilder`** interface is implemented by **`com.sun.jersey.api.client.WebResource`**, **`com.sun.jersey.api.client.AsyncWebResource`**, and other resource classes.

You can build a request using the methods defined in Table 4-3, followed by the HTTP request method, as described in How to Send HTTP Requests. Examples of how to build a request are provided in the sections that follow.

For more information about **`RequestBuilder`** and its methods, see **`http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/RequestBuilder.html`** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/Re

*() () () **Table 4-3 Building a Request***

Method	Description
`accept()`	Defines the acceptable media types. See How to Configure the Accept Header.
`acceptLanguage()`	Defines the acceptable languages using the **`acceptLanguage`** method.
`cookie()`	Adds a cookie to be set.
`entity()`	Configures the request entity. See How to Configure the Request Entity.
`header()`	Adds an HTTP header and value. See How to Configure the Accept Header.
`type()`	Configures the media type. See How to Configure the Request Entity.

() ()

How to Send HTTP Requests

Table 4-4 list the **`WebResource`** and **`AsyncWebResource`** methods that can be used to send HTTP requests.

In the case of **`AsyncWebResource`**, a **`java.util.concurrent.Future<V>`** object is returned, which can be used to access the result of the computation later, without blocking execution. For more information about Future<V>, see **`http://docs.oracle.com/javase/6/docs/api/index.html?java/util/concurrent/Future.html`** (http://docs.oracle.com/javase/6/docs/api/index.html?java/util/concurrent/Future.html).

*() () () **Table 4-4 WebResource Methods to Send HTTP Requests***

Method	Description
`get()`	Invoke the HTTP GET method to get a representation of the resource.
`post()`	Invoke the HTTP POST method to create or update the representation of the specified resource.
`put()`	Invoke the HTTP PUT method to update the representation of the resource.
`delete()`	Invoke the HTTP DELETE method to delete the representation of the resource.

If the response has an entity (or representation), then the Java type of the instance required is declared in the HTTP method.

Example 4-7 provides an example of how to send an HTTP GET request. In this example, the response entity is requested to be an instance of **`String`**. The response entity will be de-serialized to a **`String`** instance.

*() () **Example 4-7 Sending an HTTP GET Request***

Download


```
import com.sun.jersey.api.client.WebResource;
...
    public static void main(String[] args) {
...
         WebResource resource = c.resource("http://example.com/helloWorld");
         String response = resource.get(String.class);
...
```

Example 4-8 provides an example of how to send an HTTP PUT request and put the entity **foo:bar** into the Web resource. In this example, the response entity is requested to be an instance of **com.sun.jersey.api.client.ClientResponse**.

() () ***Example 4-8 Sending an HTTP PUT Request***

```
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.ClientResponse;
...
    public static void main(String[] args) {
...
         WebResource resource = c.resource("http://example.com/helloWorld");
         ClientResponse response = resource.put(ClientResponse.class, "foo:bar");
...
```

If you wish to send an HTTP request using a generic type, to avoid type erasure at runtime, you need to create a **com.sun.jersey.api.client.GenericType** object to preserve the generic type. For more information, see **http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/GenericType.html** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/Gener

Example 4-9 provides an example of how to send an HTTP request using a generic type using **GenericType** to preserve the generic type.

() () ***Example 4-9 Sending an HTTP GET Request Using a Generic Type***

```
import com.sun.jersey.api.client.WebResource;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        List<String> list = resource.get(new GenericType<List<String>>() {});
...
```

() ()

How to Pass Query Parameters

You can pass query parameters in the GET request by defining a **javax.ws.rs.core.MultivaluedMap** and using the **queryParams** method on the Web resource to pass the map as part of the HTTP request.

For more information about **MultivaluedMap**, see **http://docs.oracle.com/javaee/6/api/javax/ws/rs/core/MultivaluedMap.html** (http://docs.oracle.com/javaee/6/api/javax/ws/rs/core/MultivaluedMap.html).

Example 4-10 provides an example of how to pass parameters in a GET request to a Web resource hosted at **http://example.com/helloworld**, resulting in the following request URI: **http://example.com/base?param1=val1&param2=val2**

() () ***Example 4-10 Passing Query Parameters***

```
import com.sun.jersey.api.client.WebResource;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.MultivaluedMapImpl;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        MultivaluedMap queryParams = new MultivaluedMapImpl();
        queryParams.add("param1", "val1");
        queryParams.add("param2", "val2");
        String response = resource.queryParams(queryParams).get(String.class);
...
```

() ()

How to Configure the Accept Header

Configure the **Accept** header for the request using the **accept** method on the Web resource.

Example 4-11 provides an example of how to specify **text/plain** as the acceptable MIME media type in a GET request to a Web resource hosted at **http://example.com/helloworld**.

() () ***Example 4-11 Configuring the Accept Header***

```
import com.sun.jersey.api.client.WebResource;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        String response = resource.accept("text/plain").get(String.class);
...
```

()
()

How to Add a Custom Header

Add a custom header to the request using the **header** method on the Web resource.

Example 4-12 provides an example of how to add a custom header **FOO** with the value **BAR** in a GET request to a Web resource hosted at **http://example.com/helloworld**.

() () ***Example 4-12 Adding a Custom Header***

```
import com.sun.jersey.api.client.WebResource;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        String response = resource.header("FOO", "BAR").get(String.class);
...
```

() ()

How to Configure the Request Entity

Configure the request entity and type using the **entity** method on the Web resource. Alternatively, you can configure the request entity type only using the **type** method on the Web resource.

Example 4-13 provides an example of how to configure a request entity and type.

() () ***Example 4-13 Configuring the Request Entity***

```
import com.sun.jersey.api.client.WebResource;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        String response = resource.entity(request, MediaType.TEXT_PLAIN_TYPE).get(String.class);
...
```

Example 4-14 provides an example of how to configure the request entity media type only.

() () ***Example 4-14 Configuring the Request Entity Media Type Only***

```
import com.sun.jersey.api.client.WebResource;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        String response = resource.type(MediaType.TEXT_PLAIN_TYPE).get(String.class);
...
```

() ()

Receiving a Response from a Resource

You define the Java type of the entity (or representation) in the response when you call the HTTP method, as described in How to Send HTTP Requests.

If response metadata is required, declare the Java type **`com.sun.jersey.api.client.ClientResponse`** as the response type. the **`ClientResponse`** type enables you to access status, headers, and entity information.

The following sections describes the response metadata that you can access using the **`ClientResponse`**. For more information about **`ClientResponse`**, see **`http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/ClientResponse.html`** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/C

- How to Access the Status of Request
- How to Get the Response Entity

() ()

How to Access the Status of Request

Access the status of a client response using the **`getStatus`** method on the **`ClientResponse`** object. For a list of valid status codes, see **`http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/client/ClientResponse.Status.html`** (http://jersey.java.net/nonav/apidocs/1.9/jersey/com/sun/jersey/api/

Example 4-11 provides an example of how to access the status code of the response.

() () ***Example 4-15 Accessing the Status of the Request***

```
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.ClientResponse;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        ClientResponse response = resource.get(ClientResponse.class);
        int status = response.getStatus();
...
```

() ()

How to Get the Response Entity

Get the response entity using the **`getEntity`** method on the **`ClientResponse`** object.

Example 4-11 provides an example of how to get the response entity.

() () ***Example 4-16 Getting the Response Entity***

```
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.ClientResponse;
...
    public static void main(String[] args) {
...
        WebResource resource = c.resource("http://example.com/helloWorld");
        ClientResponse response = resource.get(ClientResponse.class);
        String entity = response.getEntity(String.class);
...
```

() ()

More Advanced RESTful Web Service Client Tasks

For more information about advanced RESTful Web service client tasks, including those listed below, see the *Jersey 1.9 User Guide* at **`http://jersey.java.net/nonav/documentation/1.9/user-guide.html`** (http://jersey.java.net/nonav/documentation/1.9/user-guide.html).

- Adding new representation types
- Using filters
- Enabling security with HTTP(s) URLConnection