### Java WebService Tutorial - Part 11 (Writing Simple REST WebService using Netbeans & GlassFish)

In this part we will see how to write a Simple REST Style Web Service using NetBeans and GlassFish Server Server and we will test the service.

**Requirements:**

1. NetBeans ID

2. GlassFish Server.

Usually NetBeans comes along with the GlassFish Server and also in build support for generating REST Services using the Jersey Framework.

So , once we have installed the NetBeans ID and GlassFish we can create our first webservice now.

**Note :** This is a quick guide of creating REST using Jersey Framework, we are not going in detail about the annotations we are using in the REST Webservice , we will discuss the basic annotation right now ,in later part we will cover in detail about each annotations with respective example.

**Step 1:**

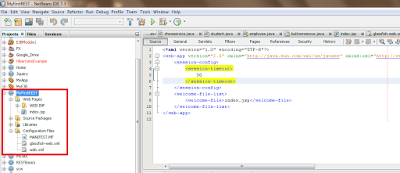
**Open NetBeans ID.**

File-->New ---> New Project --> Java Web-->Web Application and click the **"Next" Button.**

Give a name to the Project say **"MyFirstREST"** (I am using this name here for this example)and click the **"Next"** Button.

Select the Server as "**GlassFish Server"** and select the J2EE Version as **"Java EE 5"** click the **"Finish"** Button.

So , now our project folders are created .

[](http://3.bp.blogspot.com/-6ow7UGc1yKA/UgjmLZqe3MI/AAAAAAAAAHI/JDgIwi89kSg/s1600/REST-0.png)

**Step 2:**

**Creating Package.**

Right click on the Project Node "**MyFirstREST**" and select :

**New ---> Java Package** and give name as **"com.test"** (You can provide your own name for this example i am using this name)

Now we have a package where all our service class will be kept.

**Step 3:**

**Writing the REST Business Logic.**

There are couple of annotations i liked to discussed here, these are the very basic and commonly used annotations.

@Path -- Mention the Path from which you want to access a REST Service either class level or method level.

@GET -- Performs HTTP get Operation useful for getting info read only.

@Produces - This Produces the Respective output in different format such as XML,JSON,TEXT,HTML etc to the client.

**Step 4:**

**Implementing the Jersey Framework & Writing our First REST Service**

NetBeans generally comes along with the Jersey Framework if you don't have the Framework install , manually you should have to download it and add the jars to the libraries.

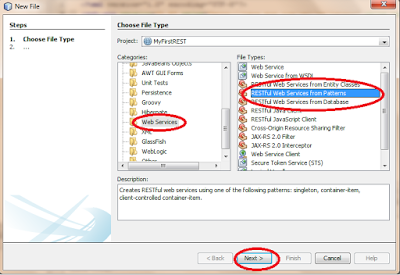
But i am not going to discuss about that , i assume you all are smart guys , and you have NetBeans with J2EE Supported version , so basically you will have Jersey supported framework install in the NetBeans Itself.

So , We don't need actually write any code from implementing this frame work in **web.xml,** NetBeans automatically does for us., that's the advantage of using NetBeans.

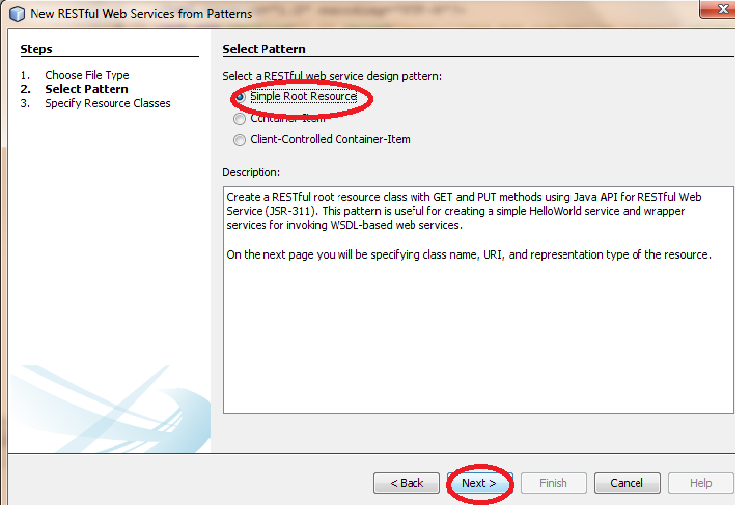
**So now this can achieve by following these steps :**

Right click on the "com.test"(Package i am using for this example, you can use your own)ans select

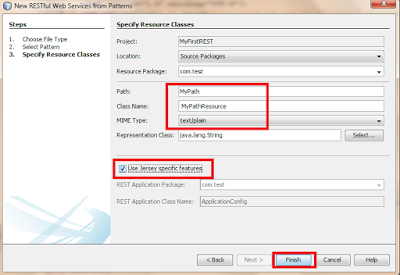
**New--->Other--->WebService-->RESTful WebServices from Patterns.**

[](http://3.bp.blogspot.com/-UyjtXHiKRdE/UgjmYM3A6kI/AAAAAAAAAHQ/nZbLM1F9FbA/s1600/REST-1.png)

**Select patterns as "Simple Root Resource" and Press the "Next" Button**

[](http://2.bp.blogspot.com/-utapGxDUKSM/Ugjmm1O8yCI/AAAAAAAAAHY/gQoyokcvf_k/s1600/REST-2.png)

**Give path Name as"MyPath" and class Name as"MypathResource" and select the MIME Type as "text/plain". and importantly select the check box for Jersey framework and click the "finish" Button.**

[](http://3.bp.blogspot.com/-bu1_iLb8KYc/Ugjm-n4g7zI/AAAAAAAAAHg/s5OJx0hKlQc/s1600/REST-3.png)

**Step 5:**

**Develop the code.**

By default a class name called "MypathResource" will be created with some methods and instance variable define in it.Delete the code and replace with the following code.

package com.test;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.UriInfo;

import javax.ws.rs.PathParam;

import javax.ws.rs.Consumes;

import javax.ws.rs.PUT;

import javax.ws.rs.Path;

import javax.ws.rs.GET;

import javax.ws.rs.Produces;

@Path("Mypath")

public class MypathResource {

@GET

@Produces("text/plain")

public String getText() {

return "My First RESTful Servivce....";

}

}

This is the very basic and simplest RESTful Service , it will accept the Path "MyPath" in the URL and invoke the method getText which in turn return a Response as Text with the wording "My First RESTful Servivce..."

**Step 6:**

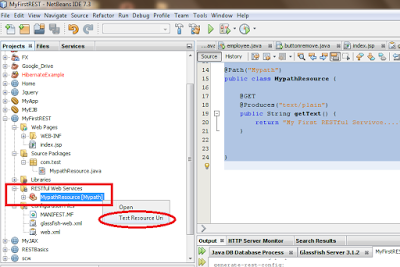
**Clean & Build , Deploy to check the service.**

Right Click on the Project Node "**MyFirstREST**" and select Clean and build this will clean the directory and compiles the Java files and creates a WAR Archive.

Once , the Clean and build is finish successfully again RightClick on the ProjectNode "**MyFirstREST"** and select **"Deploy".**

Once the Deployment is done we can check the service by:

Right click on the REST Service created under folder called **"Restful Web Service"** and select **"Test Resource Uri"**

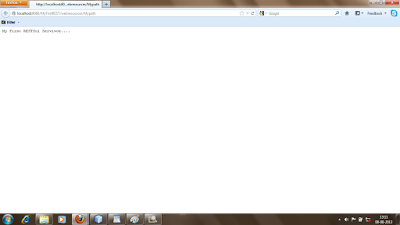
[](http://3.bp.blogspot.com/-cniVzlvJFqE/UgjnNlyTREI/AAAAAAAAAHo/RaOeJFyCb1U/s1600/REST-4.png)

**or**  we can directly access using the URL

<http://localhost:8080/MyFirstREST/webresources/Mypath>

**Note :** The Port Number will differ based upon your Server Configuration Setting.

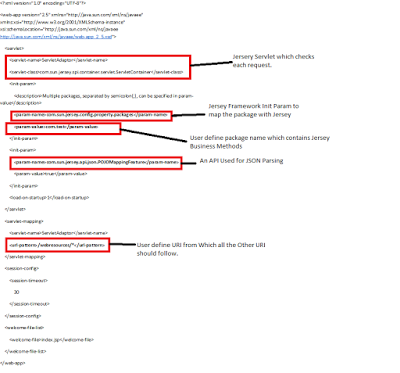
**Result :**

[](http://4.bp.blogspot.com/-onLrkvT5KOc/UgjnZFme4TI/AAAAAAAAAHw/RBHLgaJaUoM/s1600/REST-5.png)

**Main Components :**

How all these is happening ?

The main Gate Keeper or the main Servlet for this is specified in the web.xml which performs the Jersey Mechanism , Please refer the image below for detail explanation of Jersey Servlet define in web.xml .

[](http://2.bp.blogspot.com/-Td2RXJmQiDM/UgjnyF3MylI/AAAAAAAAAH4/y-KRT5Lelqk/s1600/web.png)

That's all for the day, In the Next Section we will be seeing different useful annotations and how to produce different form of Outputs.

**Thanks for Reading.**

**Please provide your valuable suggestion and comments**

### REST Annotations & Classes:

Hi, in this Section we will be discussing the different types of Annotations and some of the mostly used classes, provided by REST which can we use in our code.

First off, the most important Points we need to remember is:

REST is Web Service which performs all its Operation based upon the HTTP Methods. So it provides the following annotations.

**@ GET**

**@POST**

**@PUT**

**@DELETE**

**@HEAD**

If any one similar with the HTTP methods, they same they behave also here.

Some of the mostly used Annotations:

**@Produces** - This one we already discuss in our previous Sections, Any way it is used to produce a response to the User in based upon different MIME Types.( ex: text/html )

**@Consumes** - It is used to define what type of MIME or Inputs it will accepts from the Client .ex: forms--URL--encoded.

**@Context** - It like the ServletContext in Servlet , it is the Jersey Framework context. It can used in many cases such as to get Header Parameters, Query parameters, Form Parameters etc.

**Accessing Parameters in REST :**

REST provides the following ways, the Param can be represented.

1.@PathParam

2.@QueryParam

3.@FormParam

4.@MatrixParam

**1.@PathParam:**

This annotation is used to get the Parameter specified in the URI using {} either from class level or Method Level.

**Ex :CLASS LEVEL**

@Path("/MyPath/{username}")

class MyPathResource

{

@GET

@Produces("text/plain")

public String getText(@PathParam("username"}String username)

{

return "UserName:"+username;

}

}

**Access from URL :**

[http://localhost:8080/MyPath/**Ayaz**](http://localhost:8080/MyPath/Ayaz)

**Ayaz** - Here taken as value for "username" and map to the Methods getText.Since getText does not have any path Associated with it the Jersey will automatically invokes this method.

**Note:** If there are more than one method specifies in the class with no path Annotation, then class will compile fine but the deployment will be fail.

**EX: METHOD LEVEL**

@Path("/MyPath/{username}")

class MyPathResource

{

@GET

@Produces("text/plain")

public String getText(@PathParam("username"}String username)

{

return "UserName:"+username;

}

@Path("/getText1/{text}")

@GET

@Produces("text/plain")

public String getText1(@PathParam("text"}String text)

{

return "Enter text:"+text;

}

}

**Access from URL :**

[http://localhost:8080/MyPath/**Ayaz**](http://localhost:8080/MyPath/Ayaz) ---> Gives Output as UserName : Ayaz

[http://localhost:8080/MyPath/**getText1**/**HelloWorld**](http://localhost:8080/MyPath/getText1/HelloWorld) ---> Gives output as Enter text : HelloWorld

**Note:** If no value is pass for name error will be thrown at runtime.

**2. @Queryparam:**

Queryparam is used to access the name and value pair enter in the URL using **"?"** .

It can be defined at class level and Method level; mostly it will be used in Method level. We will discuss Method level only.

**Ex:**

[http://localhost:8080/MyPath**/MyMethod?name=ayaz&age=23**](http://localhost:8080/MyPath/MyMethod?name=ayaz&age=23)

Here , name and age are the two Query parameters.

**Ex:**

@Path("/MyMethod")

@GET

@Produces(MediaType.TEXT\_PLAIN)

public String Query(@QueryParam("name")String name ,

@QueryParam("age")String age )

{

return "Query Parameters"+" "+"Name:"+name+" "+"Age:"+age;

}

**Note:** If no value is pass for name error will be thrown at runtime.

**3. @FormParam :**

Form param is used to obtain values of the Form elements.

**Ex:**

@Path("/posted")

@POST

@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)

public String putJson(@FormParam("name")String name ) {

return name;

}

**Here, name is the name of the text field or some other field declared inside the Form tag.**

**Note:** If no value is pass for name null will be return.

**4. @MatrixParam :**

The Matrix Param is used to accept values in name & value pair unlike Query Parameter it does not need any questionmark to begin with nor any and (&)sign to specify multiple values. Multiple values can be represented using semicolon(;)

**Ex:**

[http://localhost:8080/MyPath**/MyMethod;name=ayaz;age=23**](http://localhost:8080/MyPath/MyMethod;name=ayaz;age=23)

@Path("/MyMethod")

@GET

@Produces(MediaType.TEXT\_PLAIN)

public String putMatrix(@MatrixParam("name")String name,@MatrixParam("age")int age)

{

return "Name:"+name+"Age:"+age ;

}

**Note:** If no value is pass for name and age null will be return.

**Useful Classes:**

* MediaType
* Response
* JSONObject & JSONArray

**1. MediaType :**

This class can be used to represent the MIME Types in the form of Constants.

**Ex:**

@Produces (MediaType.TEXT\_PLAIN)

 Equivalent to

@Produces ("text/plain")

**2. Response:**

The Response class one of the widely used , it can be used to return response as text,images,files etc.Instead of being returning as String we should return the Response Object and let Jersey to do the remaining writing to the output stream and other stuffs.

**Writing text Response**

**Ex:**

@Path("/matrix")

@GET

@Produces(MediaType.TEXT\_PLAIN)

public Response putMatrix(@MatrixParam("name")String name,@MatrixParam("age")int age)

{

return Response.ok("Name:"+name+"Age:"+age).build();

}

 The Response.ok method takes the String as an Entity and builds it adds to the Output as response.

**Writing Image Response:**

@GET

@Path("/getData")

@Produces("image/jpg")

public Response getData()

{

File f=new File("G:\\Icons\\en.jpg");

Response.ResponseBuilder builder=Response.ok((Object)f);

builder.header("Content-Disposition","attachment; filename=\"file\_from\_server.jpg\"");

return builder.build();

}

**3.JSONObject & JSONArray :**

**The JSON Object class is used to create a list of JSON Objects.**

**Ex:**

JSONObject jSONObject=new JSONObject();

jSONObject.put("FirstName", "Jack");

jSONObject.put("LastName", "Sparrow");

jSONObject.put("Address", "America");

System.out.println(jSONObject);

**Output :**

    {"FirstName":"Jack","LastName":"Sparrow","Address":"America"}

**The JSON Array class is used to create an Array of JSON Objects.**

**Ex:**

JSONObject jSONObject=new JSONObject();

jSONObject.put("FirstName", "Jack");

jSONObject.put("LastName", "Sparrow");

jSONObject.put("Address", "America");

JSONArray array=new JSONArray();

array.put(jSONObject);

System.out.println(array);

**Output:** [{"FirstName":"Jack","LastName":"Sparrow","Address":"America"}]

Even though Jersey with JAXB, support automatic conversion or implementation of JSON and XML from simple Java Bean with setter/getters, It is good to know how to do it manually

### Java WebService Tutorial - Part 13 ( XML & JSON )

**Producing XML and JSON:**

In the previous section we have seen how to access Parameters from URL and what are useful classes we can use in our REST Services. In this section we will see how to Produce different Responses such as XML and JSON in REST.

So far in our all example, we have used GET and access directly from the Browser URL.What if some one wants to POST data , in this case i need to used POST Method, let see the following example here.

@POST

@Path("/Postme")

@Produces("text/plain")

public Response getData(@FormParam("user")String user)

{

return Response.ok(user).build();

}

I cannot go directly in to the Browser and hit the URL with the Path .../Postme it will give me an error saying Method not allowed, we cannot directly do that, either we need a form to Post these changes and we need to write a client class to access this service. How to write a Client to access REST Service will be cover in later Parts. Now we will see how to do this by using a Html Form Submission.

**Ex:**

<form method="post" action="/....../Postme">

User :<input type="text" name="user">

<input type="submit">

</form>

When the submit button is pressed the "/Postme"Path 's respective Method get's invoked and it takes the parameter "user" which is a textfield inside the form tag ,and return the Response as the value entered in the html textfield.

As , for Producing JSON and XML we can use get or post it depend upon the criteria, now for our understanding , i am using GET to Produce JSON and XML Data and display it to the User.

**1. XML :**

XML can be produce in two ways , either using the JAXB if you don't know JAXB , no need to Panic it just an API Provided by J2EE for quickly forming XML ,Parsing XML and Producing XML.

It takes some little Annotations that's it it wont take much time to understand the basics of JAXB.

Jersey , automatically does the XML /JSON Conversion, while returning response.

Another way to produce the XML is by returning as String in the Response, it is not the best way , but it can also produce XML Output.

**EX:**

@Path("/check/{username}")

@GET

@Produces("text/xml")

public String getText(@PathParam("username")String username) {

return "<?xml version=\"1.0\" encoding=\"UTF-8\"?><root> <username>"+username+"</username></root>";

}

This code will Produce the output as String in the XML Format.

 If you access from URL: ....../check/John

**Output** :<root><username>John</username></root>

**Using JAXB Style:**

Create a Class called "Student".

package test;

import javax.xml.bind.annotation.XmlRootElement;

**@XmlRootElement // This does everthing for Creating XML .**

public class student {

private int rollno;

private String name;

public int getRollno() {

return rollno;

}

public void setRollno(int rollno) {

this.rollno = rollno;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**Inside the REST Service Class:**

@Path("/getXML")

@GET

@Produces(MediaType.APPLICATION\_XML)

public student getXML()

{

student s=new student();

s.setName("Ayaz");

s.setRollno(1440);

return s;

}

Deploy the Service & Access this Method ....../getXML will return the following Output:

**XML Output :**

<student><name>Ayaz</name><rollno>1440</rollno></student>

**Note : Just By changing MediaType.APPLICATION\_XML to**

**MediaType.APPLICATION\_JSON in the @Produces Annotation of the Method getXML will give the following JSON Output**

**JSON Output :**

{"rollno":1440,"name":"Ayaz"}

**2. JSON:**

We have already seen the how to produce JSON output using JAXB, Now we will look at how we can do it manually using the JSON classes provided by the Jackson framework which comes along with the Jersey.

It can be done by specifying the following in the web.xml file:

<init-param>

**<param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>**

**<param-value>true</param-value>**

</init-param>

This enables the Jackson framework which comes along with the Jersey.Jackson used to provide the libraries to use for accessing and producing JSON.

Ex:

@Path(“/Mydata”)

@GET

@Produces("application/json")

public Response getData() throws JSONException

{

JSONObject jSONObject=new JSONObject();

JSONArray array=new JSONArray();

for(int i=0;i<5;i++)

{

jSONObject.put("name"+i, i);

}

array.put(jSONObject);

String h=array.toString();

Response r=null;

r=Response.ok(h).build();

return r;

}

**Output:**

[{"name0":0,"name1":1,"name2":2,"name3":3,"name4":4}]

That's all folks , in the next section we will see how to write a REST client