What is Web Service

Web service is a technology to communicate one programming language with another. For example, java programming language can interact with PHP and .Net by using web services.

What is Web Service

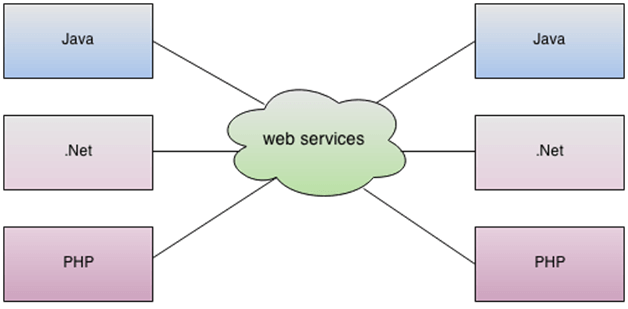
A Web Service is can be defined by following ways:

o It is a client-server application or application component for communication.

o The method of communication between two devices over the network.

o It is a software system for the interoperable machine to machine communication.

o It is a collection of standards or protocols for exchanging information between two devices or application.



As you can see in the figure, Java, .net, and PHP applications can communicate with other applications through web service over the network by using http protocals. For example, the Java application can interact with Java, .Net, and PHP applications. So web service is a language independent way of communication.

## **Types of Web Services**

1. SOAP web services.
2. RESTful web services.

SOAP WEB SERVICES:

# Web Service Components

There are three major web service components.

1. SOAP
2. WSDL
3. UDDI

## **SOAP**

SOAP is an acronym for Simple Object Access Protocol.

SOAP is a XML-based protocol for accessing web services.

SOAP is a W3C recommendation for communication between applications.

31.4M

634

HTML Tutorial

SOAP is XML based, so it is platform independent and language independent. In other words, it can be used with Java, .Net or PHP language on any platform.

## **WSDL**

WSDL is an acronym for Web Services Description Language.

WSDL is a xml document containing information about web services such as method name, method parameter and how to access it.

WSDL is a part of UDDI. It acts as a interface between web service applications.

WSDL is pronounced as wiz-dull.

## **UDDI**

UDDI is an acronym for Universal Description, Discovery and Integration.

UDDI is a XML based framework for describing, discovering and integrating web services.

UDDI is a directory of web service interfaces described by WSDL, containing information about web services.

# SOAP Web Services

SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.

SOAP is a W3C recommendation for communication between two applications.

SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

## **Advantages of Soap Web Services**

**WS Security**: SOAP defines its own security known as WS Security.

**Language and Platform independent**: SOAP web services can be written in any programming language and executed in any platform.

## **Disadvantages of Soap Web Services**

**Slow**: SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.

**WSDL dependent**: SOAP uses WSDL and doesn't have any other mechanism to discover the service.

RESTful Web Services:

REST stands for REpresentational State Transfer.

REST is an architectural style not a protocol.

Advantages of RESTful Web Services

Fast: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.

Language and Platform independent: RESTful web services can be written in any programming language and executed in any platform.

## **Java Web Services API**

There are two main API's defined by Java for developing web service applications since JavaEE 6.

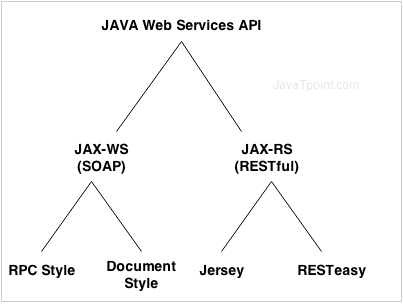
1) **JAX-WS**: for SOAP web services. The are two ways to write JAX-WS application code: by RPC style and Document style.

39.8M

654

Hello Java Program for Beginners

2) **JAX-RS**: for RESTful web services. There are mainly 2 implementation currently in use for creating JAX-RS application: Jersey and RESTeasy.

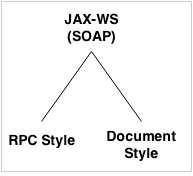


# JAX-WS Tutorial

**JAX-WS tutorial** is provides concepts and examples of JAX-WS API. This JAX-WS tutorial is designed for beginners and professionals.

There are two ways to develop JAX-WS example.

1. RPC style
2. Document style



# JAX-WS Example RPC Style

Creating JAX-WS example is a easy task because it requires no extra configuration settings.

JAX-WS API is inbuilt in JDK, so you don't need to load any extra jar file for it. Let's see a simple example of JAX-WS example in RPC style.

There are created 4 files for hello world JAX-WS example:

1. HelloWorld.java
2. HelloWorldImpl.java
3. Publisher.java
4. HelloWorldClient.java

The first 3 files are created for server side and 1 application for client side.

## **JAX-WS Server Code**

*File: HelloWorld.java*

1. **package** com.javatpoint;
2. **import** javax.jws.WebMethod;
3. **import** javax.jws.WebService;
4. **import** javax.jws.soap.SOAPBinding;
5. **import** javax.jws.soap.SOAPBinding.Style;
6. //Service Endpoint Interface
7. @WebService
8. @SOAPBinding(style = Style.RPC)
9. **public** **interface** HelloWorld{
10. @WebMethod String getHelloWorldAsString(String name);
11. }

*File: HelloWorldImpl.java*

1. **package** com.javatpoint;
2. **import** javax.jws.WebService;
3. //Service Implementation
4. @WebService(endpointInterface = "com.javatpoint.HelloWorld")
5. **public** **class** HelloWorldImpl **implements** HelloWorld{
6. @Override
7. **public** String getHelloWorldAsString(String name) {
8. **return** "Hello World JAX-WS " + name;
9. }
10. }

*File: Publisher.java*

1. **package** com.javatpoint;
2. **import** javax.xml.ws.Endpoint;
3. //Endpoint publisher
4. **public** **class** HelloWorldPublisher{
5. **public** **static** **void** main(String[] args) {
6. Endpoint.publish("http://localhost:7779/ws/hello", **new** HelloWorldImpl());
7. }
8. }

## **How to view generated WSDL**

After running the publisher code, you can see the generated WSDL file by visiting the URL:

1. http://localhost:7779/ws/hello?wsdl

## **JAX-WS Client Code**

*File: HelloWorldClient.java*

1. **package** com.javatpoint;
2. **import** java.net.URL;
3. **import** javax.xml.namespace.QName;
4. **import** javax.xml.ws.Service;
5. **public** **class** HelloWorldClient{
6. **public** **static** **void** main(String[] args) **throws** Exception {
7. URL url = **new** URL("http://localhost:7779/ws/hello?wsdl");
9. //1st argument service URI, refer to wsdl document above
10. //2nd argument is service name, refer to wsdl document above
11. QName qname = **new** QName("http://javatpoint.com/", "HelloWorldImplService");
12. Service service = Service.create(url, qname);
13. HelloWorld hello = service.getPort(HelloWorld.**class**);
14. System.out.println(hello.getHelloWorldAsString("javatpoint rpc"));
15. }
16. }

Output:

Hello World JAX-WS javatpoint rpc

# JAX-WS Example Document Style

Like RPC style, we can create JAX-WS example in document style. To do so, you need to change only one line in service interface.

You need to use Style.DOCUMENT for @SOAPBinding annotation in place of Style.RPC. Let's have a quick look at this:

*File: HelloWorld.java*

1. @SOAPBinding(style = Style.DOCUMENT)//It is changed from RPC to DOCUMENT

Now Let's see the full example of JAX-WS in document style.

27.2M

498

Prime Ministers of India | List of Prime Minister of India (1947-2020)

There are created 4 files for hello world JAX-WS document style example:

1. HelloWorld.java
2. HelloWorldImpl.java
3. Publisher.java
4. HelloWorldClient.java

The first 3 files are created for server side and 1 application for client side.

## **JAX-WS Server Code**

*File: HelloWorld.java*

1. **package** com.javatpoint;
2. **import** javax.jws.WebMethod;
3. **import** javax.jws.WebService;
4. **import** javax.jws.soap.SOAPBinding;
5. **import** javax.jws.soap.SOAPBinding.Style;
6. //Service Endpoint Interface
7. @WebService
8. @SOAPBinding(style = Style.DOCUMENT)
9. **public** **interface** HelloWorld{
10. @WebMethod String getHelloWorldAsString(String name);
11. }

*File: HelloWorldImpl.java*

1. **package** com.javatpoint;
2. **import** javax.jws.WebService;
3. //Service Implementation
4. @WebService(endpointInterface = "com.javatpoint.HelloWorld")
5. **public** **class** HelloWorldImpl **implements** HelloWorld{
6. @Override
7. **public** String getHelloWorldAsString(String name) {
8. **return** "Hello World JAX-WS " + name;
9. }
10. }

*File: Publisher.java*

1. **package** com.javatpoint;
2. **import** javax.xml.ws.Endpoint;
3. //Endpoint publisher
4. **public** **class** HelloWorldPublisher{
5. **public** **static** **void** main(String[] args) {
6. Endpoint.publish("http://localhost:7779/ws/hello", **new** HelloWorldImpl());
7. }
8. }

If you run the publisher class, it may generate following **error:**

Wrapper class com.javatpoint.GetHelloWorldAsString is not found.

Have you run APT to generate them?

To solve the problem, go to bin directory of your current project in command prompt and run the following **command:**

wsgen -keep -cp . com.javatpoint.HelloWorldImpl

Now, it will generator 2 files:

* SayHello
* SayHelloResponse

Paste these files in com.javatpoint directory and then run the publisher class.

## **How to view generated WSDL**

After running the publisher code, you can see the generated WSDL file by visiting the URL:

1. http://localhost:7779/ws/hello?wsdl

## **JAX-WS Client Code**

*File: HelloWorldClient.java*

1. **package** com.javatpoint;
2. **import** java.net.URL;
3. **import** javax.xml.namespace.QName;
4. **import** javax.xml.ws.Service;
5. **public** **class** HelloWorldClient{
6. **public** **static** **void** main(String[] args) **throws** Exception {
7. URL url = **new** URL("http://localhost:7779/ws/hello?wsdl");
9. //1st argument service URI, refer to wsdl document above
10. //2nd argument is service name, refer to wsdl document above
11. QName qname = **new** QName("http://javatpoint.com/", "HelloWorldImplService");
12. Service service = Service.create(url, qname);
13. HelloWorld hello = service.getPort(HelloWorld.**class**);
14. System.out.println(hello.getHelloWorldAsString("javatpoint document"));
15. }
16. }

Output:

Hello World JAX-WS javatpoint document

Restful Webservices:

1. POST – This would be used to create a new employee using the RESTful web service

2. GET – This would be used to get a list of all employee using the RESTful web service

3. PUT – This would be used to update all employee using the RESTful web service

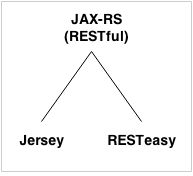
4. DELETE – This would be used to delete all employee using the RESTful services

# JAX-RS Tutorial

**JAX-RS tutorial** is provides concepts and examples of JAX-RS API. This JAX-RS tutorial is designed for beginners and professionals.

There are two main implementation of JAX-RS API.

1. Jersey
2. RESTEasy



# JAX-RS Example Jersey

We can create JAX-RS example by jersey implementation. To do so, you need to load jersey jar files or use maven framework.

In this example, we are using jersey jar files for using jersey example for JAX-RS.

[Click me to download jersey jar files.](https://static.javatpoint.com/webservicepages/download/jerseyjars.zip)

There are created 4 files for hello world JAX-RS example:

1. Hello.java
2. web.xml
3. index.html
4. HelloWorldClient.java

The first 3 files are created for server side and 1 application for client side.

Keep Watching

## **JAX-RS Server Code**

*File: Hello.java*

1. **package** com.javatpoint.rest;
2. **import** javax.ws.rs.GET;
3. **import** javax.ws.rs.Path;
4. **import** javax.ws.rs.Produces;
5. **import** javax.ws.rs.core.MediaType;
6. @Path("/hello")
7. **public** **class** Hello {
8. // This method is called if HTML and XML is not requested
9. @GET
10. @Produces(MediaType.TEXT\_PLAIN)
11. **public** String sayPlainTextHello() {
12. **return** "Hello Jersey Plain";
13. }
14. // This method is called if XML is requested
15. @GET
16. @Produces(MediaType.TEXT\_XML)
17. **public** String sayXMLHello() {
18. **return** "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
19. }
21. // This method is called if HTML is requested
22. @GET
23. @Produces(MediaType.TEXT\_HTML)
24. **public** String sayHtmlHello() {
25. **return** "<html> " + "<title>" + "Hello Jersey" + "</title>"
26. + "<body><h1>" + "Hello Jersey HTML" + "</h1></body>" + "</html> ";
27. }
28. }

*File: web.xml*

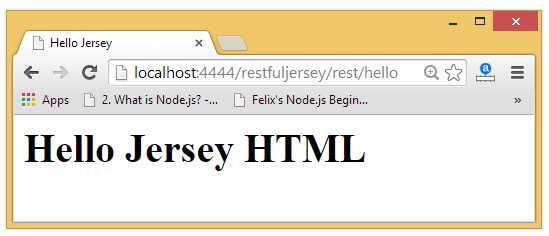
1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5. http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"
6. id="WebApp\_ID" version="3.0"**>**
7. **<servlet>**
8. **<servlet-name>**Jersey REST Service**</servlet-name>**
9. **<servlet-class>**org.glassfish.jersey.servlet.ServletContainer**</servlet-class>**
10. **<init-param>**
11. **<param-name>**jersey.config.server.provider.packages**</param-name>**
12. **<param-value>**com.javatpoint.rest**</param-value>**
13. **</init-param>**
14. **<load-on-startup>**1**</load-on-startup>**
15. **</servlet>**
16. **<servlet-mapping>**
17. **<servlet-name>**Jersey REST Service**</servlet-name>**
18. **<url-pattern>**/rest/\***</url-pattern>**
19. **</servlet-mapping>**
20. **</web-app>**

*File: index.html*

1. **<a** href="rest/hello"**>**Click Here**</a>**

Now run this application on server. Here we are using Tomcat server on port 4444. The project name is restfuljersey.

After running the project, you will see the following output:



## **JAX-RS Client Code**

The ClientTest.java file is created inside the server application. But you can run client code by other application also by having service interface and jersey jar file.

*File: ClientTest.java*

1. **package** com.javatpoint.restclient;
2. **import** java.net.URI;
3. **import** javax.ws.rs.client.Client;
4. **import** javax.ws.rs.client.ClientBuilder;
5. **import** javax.ws.rs.client.WebTarget;
6. **import** javax.ws.rs.core.MediaType;
7. **import** javax.ws.rs.core.UriBuilder;
8. **import** org.glassfish.jersey.client.ClientConfig;
9. **public** **class** ClientTest {
10. **public** **static** **void** main(String[] args) {
11. ClientConfig config = **new** ClientConfig();
12. Client client = ClientBuilder.newClient(config);
13. WebTarget target = client.target(getBaseURI());
14. //Now printing the server code of different media type
15. System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT\_PLAIN).get(String.**class**));
16. System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT\_XML).get(String.**class**));
17. System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT\_HTML).get(String.**class**));
18. }
19. **private** **static** URI getBaseURI() {
20. //here server is running on 4444 port number and project name is restfuljersey
21. **return** UriBuilder.fromUri("http://localhost:4444/restfuljersey").build();
22. }
23. }

Output:

Hello Jersey Plain

<?xml version="1.0"?><hello> Hello Jersey</hello>

<html> <title>Hello Jersey</title><body><h1>Hello Jersey HTML</h1></body></html>

# RESTful JAX-RS Annotations Example

JAX-RS API provides following annotations to develop RESTful applications in java. We are using jersey implementation for developing JAX-RS examples.

[Click me to download jersey jar files.](https://static.javatpoint.com/webservicepages/download/jerseyjars.zip)

## **JAX-RS Annotations**

The **javax.ws.rs** package contains JAX-RS annotations.

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| Path | It identifies the URI path. It can be specified on class or method. |
| PathParam | represents the parameter of the URI path. |
| GET | specifies method responds to GET request. |
| POST | specifies method responds to POST request. |
| PUT | specifies method responds to PUT request. |
| HEAD | specifies method responds to HEAD request. |
| DELETE | specifies method responds to DELETE request. |
| OPTIONS | specifies method responds to OPTIONS request. |
| FormParam | represents the parameter of the form. |
| QueryParam | represents the parameter of the query string of an URL. |
| HeaderParam | represents the parameter of the header. |
| CookieParam | represents the parameter of the cookie. |
| Produces | defines media type for the response such as XML, PLAIN, JSON etc. It defines the media type that the methods of a resource class or MessageBodyWriter can produce. |
| Consumes | It defines the media type that the methods of a resource class or MessageBodyReader can produce. |

## **JAX-RS @Path, @GET and @PathParam Annotations**

*File: HelloService.java*

1. **package** com.javatpoint.rest;
2. **import** javax.ws.rs.GET;
3. **import** javax.ws.rs.Path;
4. **import** javax.ws.rs.PathParam;
5. **import** javax.ws.rs.core.Response;
6. @Path("/hello")
7. **public** **class** HelloService{
8. @GET
9. @Path("/{param}")
10. **public** Response getMsg(@PathParam("param") String msg) {
11. String output = "Jersey say : " + msg;
12. **return** Response.status(200).entity(output).build();
13. }
14. }

*File: web.xml*

27.1M

528

History of Java

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5. http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"
6. id="WebApp\_ID" version="3.0"**>**
7. **<servlet>**
8. **<servlet-name>**Jersey REST Service**</servlet-name>**
9. **<servlet-class>**org.glassfish.jersey.servlet.ServletContainer**</servlet-class>**
10. **<init-param>**
11. **<param-name>**jersey.config.server.provider.packages**</param-name>**
12. **<param-value>**com.javatpoint.rest**</param-value>**
13. **</init-param>**
14. **<load-on-startup>**1**</load-on-startup>**
15. **</servlet>**
16. **<servlet-mapping>**
17. **<servlet-name>**Jersey REST Service**</servlet-name>**
18. **<url-pattern>**/rest/\***</url-pattern>**
19. **</servlet-mapping>**
20. **</web-app>**

*File: index.html*

1. **<a** href="rest/hello/javatpoint"**>**Click Here**</a>**

Now run this application on server, you will see the following output:

Output:

Jersey say : javatpoint

[Click me to download this example](https://static.javatpoint.com/webservicepages/download/restfuljerseypath.zip)

## **JAX-RS Multiple @PathParam Annotation**

*File: HelloService.java*

1. **package** com.javatpoint.rest;
2. **import** javax.ws.rs.GET;
3. **import** javax.ws.rs.Path;
4. **import** javax.ws.rs.PathParam;
5. **import** javax.ws.rs.core.Response;
6. @Path("/hello")
7. **public** **class** HelloService{
8. @GET
9. @Path("{year}/{month}/{day}")
10. **public** Response getDate(
11. @PathParam("year") **int** year,
12. @PathParam("month") **int** month,
13. @PathParam("day") **int** day) {
15. String date = year + "/" + month + "/" + day;
17. **return** Response.status(200)
18. .entity("getDate is called, year/month/day : " + date)
19. .build();
20. }
21. }

*File: web.xml*

It is same as above example.

*File: index.html*

1. **<a** href="rest/hello/2014/12/05"**>**Click Here**</a>**

Now run this application on server, you will see the following output:

Output:

getDate is called, year/month/day : 2014/12/5

[Click me to download this example](https://static.javatpoint.com/webservicepages/download/restfuljerseypathparam.zip)

## **JAX-RS @FormParam and @POST Annotation**

*File: HelloService.java*

1. **package** com.javatpoint.rest;
2. **import** javax.ws.rs.FormParam;
3. **import** javax.ws.rs.POST;
4. **import** javax.ws.rs.Path;
5. **import** javax.ws.rs.core.Response;
6. @Path("/product")
7. **public** **class** ProductService{
8. @POST
9. @Path("/add")
10. **public** Response addUser(
11. @FormParam("id") **int** id,
12. @FormParam("name") String name,
13. @FormParam("price") **float** price) {
15. **return** Response.status(200)
16. .entity(" Product added successfuly!<br> Id: "+id+"<br> Name: " + name+"<br> Price: "+price)
17. .build();
18. }
19. }

*File: web.xml*

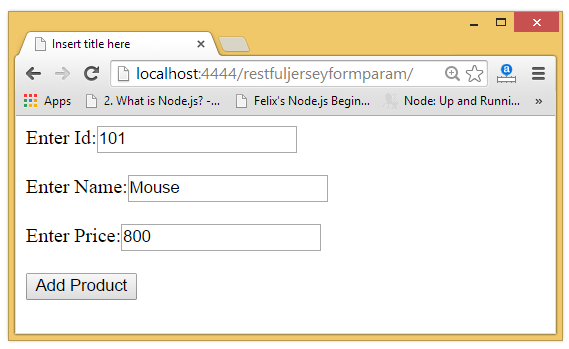
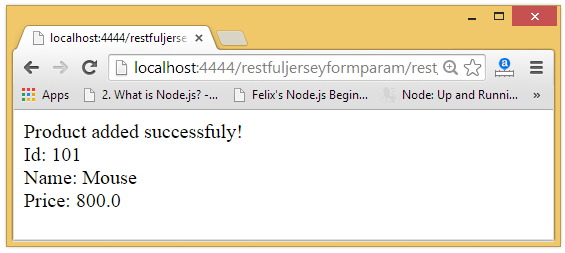
It is same as above example.

*File: index.html*

1. **<form** action="rest/product/add" method="post"**>**
2. Enter Id:**<input** type="text" name="id"**/><br/><br/>**
3. Enter Name:**<input** type="text" name="name"**/><br/><br/>**
4. Enter Price:**<input** type="text" name="price"**/><br/><br/>**
5. **<input** type="submit" value="Add Product"**/>**
6. **</form>**

Now run this application on server, you will see the following output:

Output:

# RESTful JAX-RS File Download Example

We can download text files, image files, pdf files, excel files in java by JAX-RS API. To do so we need to write few lines of code only. Here, we are using jersey implementation for developing JAX-RS file download examples.

You need to specify different content type to download different files. The @Produces annotation is used to specify the type of file content.

1. **@Produces("text/plain")**: for downloading text file.
2. **@Produces("image/png")**: for downloading png image file.
3. **@Produces("application/pdf")**: for downloading PDF file.
4. **@Produces("application/vnd.ms-excel")**: for downloading excel file.
5. **@Produces("application/msword")**: for downloading ms word file.

[Click me to download jersey jar files.](https://static.javatpoint.com/webservicepages/download/jerseyjars.zip)

## **JAX-RS Text File Download**

*File: FileDownloadService.java*

1. **package** com.javatpoint.rest;
2. **import** java.io.File;
3. **import** javax.ws.rs.GET;
4. **import** javax.ws.rs.Path;
5. **import** javax.ws.rs.Produces;
6. **import** javax.ws.rs.core.Response;
7. **import** javax.ws.rs.core.Response.ResponseBuilder;
8. @Path("/files")
9. **public** **class** FileDownloadService {
10. **private** **static** **final** String FILE\_PATH = "c:\\myfile.txt";
11. @GET
12. @Path("/txt")
13. @Produces("text/plain")
14. **public** Response getFile() {
15. File file = **new** File(FILE\_PATH);
17. ResponseBuilder response = Response.ok((Object) file);
18. response.header("Content-Disposition","attachment; filename=\"javatpoint\_file.txt\"");
19. **return** response.build();
21. }
22. }

*File: web.xml*

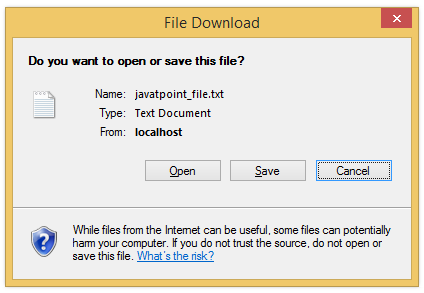
1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5. http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"
6. id="WebApp\_ID" version="3.0"**>**
7. **<servlet>**
8. **<servlet-name>**Jersey REST Service**</servlet-name>**
9. **<servlet-class>**org.glassfish.jersey.servlet.ServletContainer**</servlet-class>**
10. **<init-param>**
11. **<param-name>**jersey.config.server.provider.packages**</param-name>**
12. **<param-value>**com.javatpoint.rest**</param-value>**
13. **</init-param>**
14. **<load-on-startup>**1**</load-on-startup>**
15. **</servlet>**
16. **<servlet-mapping>**
17. **<servlet-name>**Jersey REST Service**</servlet-name>**
18. **<url-pattern>**/rest/\***</url-pattern>**
19. **</servlet-mapping>**
20. **</web-app>**

*File: index.html*

1. **<a** href="rest/files/txt"**>**Download Text File**</a>**

Now run this application on server, you will see the following output:

Output:

  
[Click me to download this example](https://static.javatpoint.com/webservicepages/download/restfuljerseyfiledownload.zip)

## **JAX-RS Image File Download**

*File: FileDownloadService.java*

1. **package** com.javatpoint.rest;
2. **import** java.io.File;
3. **import** javax.ws.rs.GET;
4. **import** javax.ws.rs.Path;
5. **import** javax.ws.rs.Produces;
6. **import** javax.ws.rs.core.Response;
7. **import** javax.ws.rs.core.Response.ResponseBuilder;
8. @Path("/files")
9. **public** **class** FileDownloadService {
10. **private** **static** **final** String FILE\_PATH = "c:\\myimage.png";
11. @GET
12. @Path("/image")
13. @Produces("image/png")
14. **public** Response getFile() {
15. File file = **new** File(FILE\_PATH);
16. ResponseBuilder response = Response.ok((Object) file);
17. response.header("Content-Disposition","attachment; filename=\"javatpoint\_image.png\"");
18. **return** response.build();
20. }
21. }

*File: web.xml*

Same as above example.

*File: index.html*

1. **<a** href="rest/files/image"**>**Download Image File**</a>**

[Click me to download this example](https://static.javatpoint.com/webservicepages/download/restfuljerseyfiledownloadimage.zip)

## **JAX-RS PDF File Download**

*File: FileDownloadService.java*

1. **package** com.javatpoint.rest;
2. **import** java.io.File;
3. **import** javax.ws.rs.GET;
4. **import** javax.ws.rs.Path;
5. **import** javax.ws.rs.Produces;
6. **import** javax.ws.rs.core.Response;
7. **import** javax.ws.rs.core.Response.ResponseBuilder;
8. @Path("/files")
9. **public** **class** FileDownloadService {
10. **private** **static** **final** String FILE\_PATH = "c:\\mypdf.pdf";
11. @GET
12. @Path("/pdf")
13. @Produces("application/pdf")
14. **public** Response getFile() {
15. File file = **new** File(FILE\_PATH);
16. ResponseBuilder response = Response.ok((Object) file);
17. response.header("Content-Disposition","attachment; filename=\"javatpoint\_pdf.pdf\"");
18. **return** response.build();
19. }
20. }

*File: web.xml*

Same as above example.

*File: index.html*

1. **<a** href="rest/files/pdf"**>**Download PDF File**</a>**

# RESTful JAX-RS File Upload Example

Like download in previous page, we can easily upload a file such as image file, pdf file, excel file, text file etc.

The @FormDataParam("file") annotation is used to mention file parameter in the service class. The @Consumes(MediaType.MULTIPART\_FORM\_DATA) is used to provide information of the file upload.

To upload file using JAX-RS API, we are using jersey implementation.

[Click me to download jersey jar files.](https://static.javatpoint.com/webservicepages/download/jerseyjars.zip)

To upload file through jersey implementation, you need to provide extra configuration entry in web.xml file.

1. **<init-param>**
2. **<param-name>**jersey.config.server.provider.classnames**</param-name>**
3. **<param-value>**org.glassfish.jersey.filter.LoggingFilter;
4. org.glassfish.jersey.media.multipart.MultiPartFeature**</param-value>**
5. **</init-param>**

Let's see the complete code to upload file using RESTful JAX-RS API.

## **JAX-RS File Upload**

*File: FileUploadService.java*

1. **package** com.javatpoint.rest;
2. **import** java.io.File;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.InputStream;
6. **import** javax.ws.rs.Consumes;
7. **import** javax.ws.rs.POST;
8. **import** javax.ws.rs.Path;
9. **import** javax.ws.rs.core.MediaType;
10. **import** javax.ws.rs.core.Response;
11. **import** org.glassfish.jersey.media.multipart.FormDataContentDisposition;
12. **import** org.glassfish.jersey.media.multipart.FormDataParam;
13. @Path("/files")
14. **public** **class** FileUploadService {
15. @POST
16. @Path("/upload")
17. @Consumes(MediaType.MULTIPART\_FORM\_DATA)
18. **public** Response uploadFile(
19. @FormDataParam("file") InputStream uploadedInputStream,
20. @FormDataParam("file") FormDataContentDisposition fileDetail) {
21. String fileLocation = "e://" + fileDetail.getFileName();
22. //saving file
23. **try** {
24. FileOutputStream out = **new** FileOutputStream(**new** File(fileLocation));
25. **int** read = 0;
26. **byte**[] bytes = **new** **byte**[1024];
27. out = **new** FileOutputStream(**new** File(fileLocation));
28. **while** ((read = uploadedInputStream.read(bytes)) != -1) {
29. out.write(bytes, 0, read);
30. }
31. out.flush();
32. out.close();
33. } **catch** (IOException e) {e.printStackTrace();}
34. String output = "File successfully uploaded to : " + fileLocation;
35. **return** Response.status(200).entity(output).build();
36. }
37. }

*File: web.xml*

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<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
3. http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
4. **<servlet>**
5. **<servlet-name>**Jersey REST Service**</servlet-name>**
6. **<servlet-class>**org.glassfish.jersey.servlet.ServletContainer**</servlet-class>**
7. **<init-param>**
8. **<param-name>**jersey.config.server.provider.packages**</param-name>**
9. **<param-value>**com.javatpoint.rest**</param-value>**
10. **</init-param>**
11. **<init-param>**
12. **<param-name>**jersey.config.server.provider.classnames**</param-name>**
13. **<param-value>**org.glassfish.jersey.filter.LoggingFilter;
14. org.glassfish.jersey.media.multipart.MultiPartFeature**</param-value>**
15. **</init-param>**
16. **<load-on-startup>**1**</load-on-startup>**
17. **</servlet>**
18. **<servlet-mapping>**
19. **<servlet-name>**Jersey REST Service**</servlet-name>**
20. **<url-pattern>**/rest/\***</url-pattern>**
21. **</servlet-mapping>**
22. **</web-app>**

*File: index.html*

1. **<h2>**File Upload Example**</h2>**
2. **<form** action="rest/files/upload" method="post" enctype="multipart/form-data"**>**
3. **<p>**
4. Select a file : **<input** type="file" name="file" size="45" **/>**
5. **</p>**
6. **<input** type="submit" value="Upload File" **/>**
7. **</form>**

Now run this application on server, you will see the following output:

Output:

1. File successfully uploaded to e://myimage.png