RESTful Web service and Jersey

By Debadatta Mishra

What is REST and what are the characteristics of RESTful web service ?

REST is an architectural style of exposing web application as a service. It was coined by Roy Fielding. REST is an acronym standing for Representational State Transfer. **REST - An Architectural Style, Not a Standard**. Because REST is just an architectural style. In REST each unique URL is a representation of some object. You can manipulate the contents of that object using an **HTTP GET,POST, PUT, or DELETE** methods.You can't bottle up that style. You can only understand it, and design your Web services in that style. The semantics of the HTTP protocol is used by REST. In REST the HTTP methods GET, PUT, POST and DELETE play a central role. They are the "verbs" that are applied to the "nouns", respectively the resources. With this limited pool of four methods all use cases have to be covered. REST web services can be developed in the following fashions . You want to develop an application for a pharmaceutical company and the wholesale dealer wants to know the products of this company. So the following url can be accessed to know the products. Let us consider the use of REST.

Example of a REST Application

As an example for a RESTful application we use an online shop. In the shop customers can fill their shopping carts with goods. Each individual object of the application is reachable over an URL as a resource. For example you can reach the shopping cart no. 5873 by the following HTTP request:

GET /shoppingcart/5873

The structure of a response is not specified in REST. There must be a common understanding about the representation between client and server. The usage of XML makes a representation for computers as well as for humans comprehensible. However REST can accept any mode of communication , it may be XML, JSOn, or Text. For example, the representation of a shopping cart can look like the following listing:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version="1.0"?>
<shoppingcart xmlns:xlink="http://www.w3.org/1999/xlink">
  <customer xlink:href="http://shop.oio.de/customer/5873">
    5873
  </customer>
  <position nr="1" amount="5">
    <article xlink:href="http://shop.oio.de/article/4501" nr="4501">
      <description>Sugar</description>
    </article>
  </position>
  <position nr="2" amount="2">
    <article xlink:href="http://shop.oio.de/article/5860" nr="5860">
      <description>Earl Grey Tea</description>
    </article>
  </position>
</shoppingcart>
```

As you can see , the server's response contains a XML document that the client can process. This document can be converted into HTML, SVG, PDF or something else by applying a XSL transformation. The shopping cart contains two positions and the associated customer.

GET /article/5860

With each document the client can enter more deeply into the web of application data. As point of entrance only one URL is sufficient. The idea of hypertext is applied to web services.

Ordering or Triggering Operations on the Server

Sugar should not be missing in a good tea. So, the order must be extended by this important article. Suppose sugar has the article number 961. The following POST request adds sugar to the shopping cart.

POST /shoppingcart/articles&cart=5873&article=961

Creating New Resources

To create new resources you can use the HTTP method PUT.

PUT /article

```
<articles>
  <name>Tea</name>
  <description>
    Herbal Tea from South Africa
  </description>
  <price>2.80</price>
  <unit>100 grams</unit>
</article>
```

The HTTP status code 201 means "created" , i.e. a new resource was generated. To indicate the location of the newly created resource a HTTP location header field is used. The client can now follow the URL to a representation of the new article.

```
HTTP/1.1 201 OK
Content-Type: text/xml;
Content-Length: 30
Location: http://shop.oio.de/article/6005
```

Deleting Resources

To delete the newly created resource, you can use a HTTP DELETE request. The following DELETE request removes the article with the id 6005.

```
DELETE /articles/6005
```

http://www.pharma.com/products

REST Web Services Characteristics

Client-Server: a pull-based interaction style: consuming components pull representations. Stateless: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. Cache: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable. Uniform interface: all resources are accessed with a generic interface (e.g., **HTTP GET, POST, PUT, DELETE**). Named resources - the system is comprised of resources which are named using a URL. Interconnected resource representations - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another. Layered components - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.

The Methods used in REST

The interface of REST is generic. There is no need for protocol conventions for the communication between client and server. The following list describes the meaning of the HTTP methods and how they are used by REST. The HTTP methods GET, POST, PUT and DELETE are a generic interface for each REST resource. Database applications have shown that a limited set of commands controlling the lifecycle of data is sufficent. The database query language SQL covers the lifecycle by the **SELECT, INSERT, UPDATE and DELETE** instructions.

Messages used in REST

All types of documents can be used as representations for resources. For example HTML, GIF and PDF files are used in the Web. XML is suitable for the transmission of structured data. If you want to realize your application by using Rest, you don't have to learn a new format. You can use well-known formats. REST messages are self explaining. In a message everything must be contained, in order to interpret the message. There is no knowledge about previous or later messages necessary for the interpretation of a message. The status of an application is represented by the contents of one or several hypertext documents.

State and Session

The server knows its state. He is not interested in the state of the client nor in sessions to its clients. The client has to manage its state itself. It decides the order in which it calls methods on the server, too. Usually in REST applications no special functionality is needed for a login. All recources can be authenticated by available web technologies, e.g. HTTP and HTTPS.

Properties of a REST Application

The REST style is characterized by the following properties: Communication takes place on call. The Client is active and requests a representation from the passive server and/or modifies a resource. A resource can be addressed by an unique URI. The client can request the representation of a resource in form of a document. Representations can refer to further resources. The server does not monitor the status of its clients. Each query to the server must contain all informations that are necessary for interpreting itself. Caching is supported. The server can mark its answers as cacheable or not cacheable.

Why is it called Representational State Transfer?

The Web is comprised of resources. A resource is any item of interest. For example, the Boeing Aircraft Corp may define a 747 resource. Clients may access that resource with this URL:

http://www.boeing.com/aircraft/747

A representation of the resource is returned (e.g., Boeing747.html). The representation places the client application in a state. The result of the client traversing a hyperlink in Boeing747.html is another resource is accessed. The new representation places the client application into yet another state. Thus, the client application changes (transfers) state with each resource representation --> Representational State Transfer!.

While REST is not a standard, it does use standards:

HTTP
URL
XML/HTML/GIF/JPEG/etc (Resource Representations)
text/xml, text/html, image/gif, image/jpeg, etc (MIME Types)

Why the name REpresentational State transfer is used, becomes clear from the following scenario. A web browser requests a website, or more generally it requests a representation of a resource using an URL. A HTML document, which represents the resource is transferred from the server to the client. The HTML document can contain links, which refer to further resources in the web. If the client navigates to a new site he changes its state. Therefore, a transfer from a state to another has happened by following a link.

Principles of REST Web Service Design

REST Web service follows four basic design principles:

Use HTTP methods explicitly.
Be stateless.
Expose directory structure-like URIs.
Transfer XML, JavaScript Object Notation (JSON), or both.

1. The key to creating Web Services in a REST network (i.e., the Web) is to identify all of the conceptual entities that you wish to expose as services. Above we saw some examples of resources: parts list, detailed part data, purchase order.

2. Create a URL to each resource.
http://www.parts-depot.com/parts/00345

3. Categorize your resources according to whether clients can just receive a representation of the resource, or whether clients can modify (add to) the resource. For the former, make those resources accessible using an HTTP GET. For the later, make those resources accessible using HTTP POST, PUT, and/or DELETE.

4. All resources accessible via HTTP GET should be side-effect free. That is, the resource should just return a representation of the resource. Invoking the resource should not result in modifying the resource.

5. Put hyperlinks within resource representations to enable clients to drill down for more information, and/or to obtain related information.

6. Design to reveal data gradually. Don't reveal everything in a single response document. Provide hyperlinks to obtain more details.

7. Specify the format of response data using a schema (DTD, W3C Schema, RelaxNG, or Schematron).
For those services that require a POST or PUT to it, also provide a schema to specify the format of the response.

8. Describe how your services are to be invoked using either a WSDL document, or simply an HTML document.

One of the key characteristics of a RESTful Web service is the explicit use of HTTP methods. REST asks developers to use HTTP methods explicitly and in a way that's consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods.

According to this mapping:

To create a resource on the server, use POST.
To retrieve a resource, use GET.
To change the state of a resource or to update it, use PUT.
To remove or delete a resource, use DELETE.

Big Players in the Web Market

All of Yahoo's web services use REST, including Flickr, del.icio.us API uses it, pubsub, bloglines, technorati, and both eBay, and Amazon have web services for both REST and SOAP. Google seams to be consistent in implementing their web services to use SOAP, with the exception of Blogger, which uses XML-RPC. You will find SOAP web services in lots of enterprise software as well.

Advantages and Disadvantages

The main advantages of REST web services are:
Lightweight - not a lot of extra xml markup
Human Readable Results
Easy to build - no toolkits required

SOAP also has some advantages:
Easy to consume - sometimes
Rigid - type checking, adheres to a contract
Development tools

Which one to use REST or WSDL based web service ?

For consuming web services, its sometimes a toss up between which is easier. For instance Google's AdWords web service is really hard to consume (in CF anyways), it uses SOAP headers, and a number of other things that make it kind of difficult. On the converse, Amazon's REST web service can sometimes be tricky to parse because it can be highly nested, and the result schema can vary quite a bit based on what you search for.

An article taken from IBM site
http://www.ibm.com/developerworks/webservices/library/ws-restful/

The basics

REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use.

REST didn't attract this much attention when it was first introduced in 2000 by Roy Fielding at the University of California, Irvine, in his academic dissertation, "Architectural Styles and the Design of Network-based Software Architectures," which analyzes a set of software architecture principles that use the Web as a platform for distributed computing (see Resources for a link to this dissertation). Now, years after its introduction, major frameworks for REST have started to appear and are still being developed because it's slated, for example, to become an integral part of Java™ 6 through JSR-311.

This article suggests that in its purest form today, when it's attracting this much attention, a concrete implementation of a **REST Web service follows four basic design principles**:

Use HTTP methods explicitly.
Be stateless.
Expose directory structure-like URIs.
Transfer XML, JavaScript Object Notation (JSON), or both.

The following sections expand on these four principles and propose a technical rationale for why they might be important for REST Web service designers.

Use HTTP methods explicitly

One of the key characteristics of a RESTful Web service is the explicit use of HTTP methods in a way that follows the protocol as defined by RFC 2616. HTTP GET, for instance, is defined as a data-producing method that's intended to be used by a client application to retrieve a resource, to fetch data from a Web server, or to execute a query with the expectation that the Web server will look for and respond with a set of matching resources. REST asks developers to use HTTP methods explicitly and in a way that's consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping:

To create a resource on the server, use POST.
To retrieve a resource, use GET.
To change the state of a resource or to update it, use PUT.
To remove or delete a resource, use DELETE.

An unfortunate design flaw inherent in many Web APIs is in the use of HTTP methods for unintended purposes. The request URI in an HTTP GET request, for example, usually identifies one specific resource. Or the query string in a request URI includes a set of parameters that defines the search criteria used by the server to find a set of matching resources. At least this is how the HTTP/1.1 RFC describes GET. But there are many cases of unattractive Web APIs that use HTTP GET to trigger something transactional on the server—for instance, to add records to a database. In these cases the GET request URI is not used properly or at least not used

RESTfully. If the Web API uses GET to invoke remote procedures, it looks like this:

GET /adduser?name=Robert HTTP/1.1

It's not a very attractive design because the Web method above supports a state-changing operation over HTTP GET. Put another way, the HTTP GET request above has side effects. If successfully processed, the result of the request is to add a new user—in this example, Robert—to the underlying data store. The problem here is mainly semantic. Web servers are designed to respond to HTTP GET requests by retrieving resources that match the path (or the query criteria) in the request URI and return these or a representation in a response, not to add a record to a database. From the standpoint of the intended use of the protocol method then, and from the standpoint of HTTP/1.1-compliant Web servers, using GET in this way is inconsistent.

Beyond the semantics, the other problem with GET is that to trigger the deletion, modification, or addition of a record in a database, or to change server-side state in some way, it invites Web caching tools (crawlers) and search engines to make server-side changes unintentionally simply by crawling a link. A simple way to overcome this common problem is to move the parameter names and values on the request URI into XML tags. The resulting tags, an XML representation of the entity to create, may be sent in the body of an HTTP POST whose request URI is the intended parent of the entity (see Listings 1 and 2).

Listing 1. Before
```
GET /adduser?name=Robert HTTP/1.1
```

Listing 2. After
```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```

The method above is exemplary of a RESTful request: proper use of HTTP POST and inclusion of the payload in the body of the request. On the receiving end, the request may be processed by adding the resource contained in the body as a subordinate of the resource identified in the request URI; in this case the new resource should be added as a child of /users. This containment relationship between the new entity and its parent, as specified in the POST request, is analogous to the way a file is subordinate to its parent directory. The client sets up the relationship between the entity and its parent and defines the new entity's URI in the POST request. A client application may then get a representation of the resource using the new URI, noting that at least logically the resource is located under /users, as shown in Listing 3.

Listing 3. HTTP GET request

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

Using GET in this way is explicit because GET is for data retrieval only. GET is an operation that should be free of side effects, a property also known as idempotence.

A similar refactoring of a Web method also needs to be applied in cases where an update operation is supported over HTTP GET, as shown in Listing 4.

Listing 4. Update over HTTP GET

```
GET /updateuser?name=Robert&newname=Bob HTTP/1.1
```

This changes the name attribute (or property) of the resource. While the query string can be used for such an operation, and Listing 4 is a simple one, this query-string-as-method-signature pattern tends to break down when used for more complex operations. Because your goal is to make explicit use of HTTP methods, a more RESTful approach is to send an HTTP PUT request to update the resource, instead of HTTP GET, for the same reasons stated above (see Listing 5).

Listing 5. HTTP PUT request

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Bob</name>
</user>
```

Using PUT to replace the original resource provides a much cleaner interface that's consistent with REST's principles and with the definition of HTTP methods. The PUT request in Listing 5 is explicit in the sense that it points at the resource to be updated by identifying it in the request URI and in the sense that it transfers a new representation of the resource from client to server in the body of a PUT request instead of transferring the resource attributes as a loose set of parameter names and values on the request URI. Listing 5 also has the effect of renaming the resource from Robert to Bob, and in doing so changes its URI to /users/Bob. In a REST Web service, subsequent requests for the resource using the old URI would generate a standard 404 Not Found error. As a general design principle, it helps to follow REST guidelines for using HTTP methods explicitly by using nouns in URIs instead of verbs. In a RESTful Web service, the verbs—POST, GET, PUT, and DELETE—are already defined by the protocol. And ideally, to keep the interface generalized and to allow clients to be explicit about the operations they invoke, the Web service should not define more verbs or remote procedures, such as /adduser or /updateuser. This general design principle also applies to the body of an HTTP request, which is intended to be used to transfer resource state, not to carry the name of a remote method or remote procedure to be invoked.

Be stateless

REST Web services need to scale to meet increasingly high performance demands. Clusters of servers with load-balancing and failover capabilities, proxies, and gateways are typically arranged in a way that forms a service topology, which allows requests to be forwarded from one server to the other as needed to decrease the overall response time of a Web service call. Using intermediary servers to improve scale requires REST Web service clients to send complete, independent requests; that is, to send requests that include all data needed to be fulfilled so that the components in the intermediary servers may forward, route, and load-balance without any state being held locally in between requests. A complete, independent request doesn't require the server, while processing the request, to retrieve any kind of application context or state. A REST Web service application (or client) includes within the HTTP headers and body of a request all of the parameters, context, and data needed by the server-side component to generate a response. Statelessness in this sense improves Web service performance and simplifies the design and implementation of server-side components because the absence of state on the server removes the need to synchronize session data with an external application. Figure 1 illustrates a stateful service from which an application may request the next page in a multipage result set, assuming that the service keeps track of where the application leaves off while navigating the set. In this stateful design, the service increments and stores a previousPage variable somewhere to be able to respond to requests for next. Stateful services like this get complicated. In a Java Platform, Enterprise Edition (Java EE) environment stateful services require a lot of up-front consideration to efficiently store and enable the synchronization of session data across a cluster of Java EE containers. In this type of environment, there's a problem familiar to servlet/JavaServer Pages (JSP) and Enterprise JavaBeans (EJB) developers who often struggle to find the root causes of java.io.NotSerializableException during session replication. Whether it's thrown by the servlet container during HttpSession replication or thrown by the EJB container during stateful EJB replication, it's a problem that can cost developers days in trying to pinpoint the one object that doesn't implement Serializable in a sometimes complex graph of objects that constitute the server's state. In addition, session synchronization adds overhead, which impacts server performance. Stateless server-side components, on the other hand, are less complicated to design, write, and distribute across load-balanced servers. A stateless service not only performs better, it shifts most of the responsibility of maintaining state to the client application. In a RESTful Web service, the server is responsible for generating responses and for providing an interface that enables the client to maintain application state on its own. For example, in the request for a multipage result set, the client should include the actual page number to retrieve instead of simply asking for next . A stateless Web service generates a response that links to the next page number in the set and lets the client do what it needs to in order to keep this value around. This aspect of RESTful Web service design can be broken down into two sets of responsibilities as a high-level separation that clarifies just how a stateless service can be maintained:

Server

Generates responses that include links to other resources to allow applications to navigate between related resources. This type of response embeds links. Similarly, if the request is for a parent or container resource, then a typical RESTful response might also include links to the parent's children or subordinate resources so that these remain connected. Generates responses that indicate whether they are cacheable or not to improve performance by reducing the number of requests for duplicate resources and by eliminating some requests entirely. The server does this by including a Cache-Control and Last-Modified (a date value) HTTP response header.

Client application

Uses the Cache-Control response header to determine whether to cache the resource (make a local copy of it) or not. The client also reads the Last-Modified response header and sends back the date value in an If-Modified-Since header to ask the server if the resource

has changed. This is called Conditional GET, and the two headers go hand in hand in that the server's response is a standard 304 code (Not Modified) and omits the actual resource requested if it has not changed since that time. A 304 HTTP response code means the client can safely use a cached, local copy of the resource representation as the most up-to-date, in effect bypassing subsequent GET requests until the resource changes. Sends complete requests that can be serviced independently of other requests. This requires the client to make full use of HTTP headers as specified by the Web service interface and to send complete representations of resources in the request body. The client sends requests that make very few assumptions about prior requests, the existence of a session on the server, the server's ability to add context to a request, or about application state that is kept in between requests. This collaboration between client application and service is essential to being stateless in a RESTful Web service. It improves performance by saving bandwidth and minimizing server-side application state.

Expose directory structure-like URIs

From the standpoint of client applications addressing resources, the URIs determine how intuitive the REST Web service is going to be and whether the service is going to be used in ways that the designers can anticipate. A third RESTful Web service characteristic is all about the URIs. REST Web service URIs should be intuitive to the point where they are easy to guess. Think of a URI as a kind of self-documenting interface that requires little, if any, explanation or reference for a developer to understand what it points to and to derive related resources. To this end, the structure of a URI should be straightforward, predictable, and easily understood. One way to achieve this level of usability is to define directory structure-like URIs. This type of URI is hierarchical, rooted at a single path, and branching from it are subpaths that expose the service's main areas. According to this definition, a URI is not merely a slash-delimited string, but rather a tree with subordinate and superordinate branches connected at nodes. For example, in a discussion threading service that gathers topics ranging from Java to paper, you might define a structured set of URIs like this: **http://www.myservice.org/discussion/topics/{topic}**

The root, /discussion, has a /topics node beneath it. Underneath that there are a series of topic names, such as gossip, technology, and so on, each of which points to a discussion thread. Within this structure, it's easy to pull up discussion threads just by typing something after /topics/.

In some cases, the path to a resource lends itself especially well to a directory-like structure. Take resources organized by date, for instance, which are a very good match for using a hierarchical syntax.

This example is intuitive because it is based on rules: **http://www.myservice.org/discussion/2008/12/10/{topic}**

The first path fragment is a four-digit year, the second path fragment is a two-digit day, and the third fragment is a two-digit month. It may seem a little silly to explain it that way, but this is the level of simplicity we're after. Humans and machines can easily generate structured URIs like this because they are based on rules. Filling in the path parts in the slots of a syntax makes them good because there is a definite pattern from which to compose them:
http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}

Some additional guidelines to make note of while thinking about URI structure for a RESTful Web service are:

Hide the server-side scripting technology file extensions (.jsp, .php, .asp), if any, so you can port to something else without changing the URIs.

Keep everything lowercase.

Substitute spaces with hyphens or underscores (one or the other).

Avoid query strings as much as you can.

Instead of using the 404 Not Found code if the request URI is for a partial path, always provide a default page or resource as a response.

URIs should also be static so that when the resource changes or the implementation of the service changes, the link stays the same. This allows bookmarking. It's also important that the relationship between resources that's encoded in the URIs remains independent of the way the relationships are represented where they are stored.

Transfer XML, JSON, or both

A resource representation typically reflects the current state of a resource, and its attributes, at the time a client application requests it. Resource representations in this sense are mere snapshots in time. This could be a thing as simple as a representation of a record in a database that consists of a mapping between column names and XML tags, where the element values in the XML contain the row values. Or, if the system has a data model, then according to this definition a resource representation is a snapshot of the attributes of one of the things in your system's data model. These are the things you want your REST Web service to serve up. The last set of constraints that goes into a RESTful Web service design has to do with the format of the data that the application and service exchange in the request/response payload or in the HTTP body. This is where it really pays to keep things simple, human-readable, and connected. The objects in your data model are usually related in some way, and the relationships between data model objects (resources) should be reflected in the way they are represented for transfer to a client application. In the discussion threading service, an example of connected resource representations might include a root discussion topic and its attributes, and embed links to the responses

given to that topic.

XML representation of a discussion thread

```
<?xml version="1.0"?>
<discussion date="{date}" topic="{topic}">
  <comment>{comment}</comment>
  <replies>
    <reply from="joe@mail.com" href="/discussion/topics/{topic}/joe"/>
    <reply from="bob@mail.com" href="/discussion/topics/{topic}/bob"/>
  </replies>
</discussion>
```

And last, to give client applications the ability to request a specific content type that's best suited for them, construct your service so that it makes use of the built-in HTTP Accept header, where the value of the header is a MIME type. Some common MIME types used by RESTful services are shown in Table 1.

Common MIME types used by RESTful services

MIME-Type	Content-Type
JSON	**application/json**
XML	**application/xml**
XHTML	**application/xhtml+xml**

This allows the service to be used by a variety of clients written in different languages running on different platforms and devices. Using MIME types and the HTTP Accept header is a mechanism known as content negotiation, which lets clients choose which data format is right for them and minimizes data coupling between the service and the applications that use it.

Conclusion

REST is not always the right choice. It has caught on as a way to design Web services with less dependence on proprietary middleware (for example, an application server) than the SOAP- and WSDL-based kind. And in a sense, REST is a return to the Web the way it was before the age of the big application server, through its emphasis on the early Internet standards, URI and HTTP. As you've examined in the so-called principles of RESTful interface design, XML over HTTP is a powerful interface that allows internal applications, such as Asynchronous JavaScript + XML (Ajax)-based custom user interfaces, to easily connect, address, and consume resources. In fact, the great fit between Ajax and REST has increased the amount of attention REST is getting these days. Exposing a system's resources through a RESTful API is a flexible way to provide different kinds of applications with data formatted in a standard way. It helps to meet integration requirements that are critical to building systems where data can be easily combined (mashups) and to extend or build on a set of base, RESTful services into something much bigger. This article touches on just the basics here but hopefully in a way that has enticed you to continue exploring the subject.

REST Webservices using Jersey from java.net

URL : http://jersey.java.net/
Latest version : 1.12

What is Jersey ?
Jersey is the open source, production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.

How will you create a RESTful Web service using Jersey ?
Capsule:
1. Download the jersey related jar files java.net sites and configure it in your lib directory. Remember that jersey depends upon servlet-api.jar file.
2. Jersey provides annotation based RESTful web service development. For this write a java class having methods which need to be exposed as REST service methods. Provide the relevent annotations.
3. First of all define your java package structure based upon the business logic and write the java classes by providing the appropriate http method, parameters and url path annotation.
4. The only configuration required for Jersey is in java packages for the Jersey Servlet in web.xml file. The example is given below.

```
<servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>com.ddlab.rnd.rest.jersy</param-value>
    </init-param>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>com.ddlab.rnd.rest.jersy.util</param-value>
    </init-param>
     <load-on-startup>1</load-on-startup>
</servlet>
```

It is also possible to add more packages.

5. Run the build script to creat the war file and deploy it in any J2EE server.
6. To know whether the RESTful application has been deployed or not, use the url for WADL.
The following example is given below.

http://localhost:8080/restfulservicejersey/application.wadl

If you are able to see the XML structure, then it is a perfect deployment.

Before creating an example let us understand some concept on creating a RESTful web service. In Jersey, each web service class can be represented as resource. In the below example, **@Path("/All")** indicates that to access or consume the methods of this class, we have to access the URL in this manner "**http://localhost:8080/restfulservicejersey/All**". So the main URL becomes "**http://localhost:8080/restfulservicejersey/All**".

The following annotations are used in the class level. The @Path annotation's value is a relative URI path. The Java class will be hosted at the URI path /helloworld. **@Path** - This annotation can be used for both class and method level.
For example , @Path("/All")
The following annotations are used in the method level.
@GET, @PUT, @POST, @DELETE,@HEAD,@Produces,@Consumes
@Produces : The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client. An example is given below.

```
@GET
@Produces({"application/xml", "application/json"})
public String doGetAsXmlOrJson()
{
...
}
```

@Consumes : The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that

were sent by the client. The above example can be modified to set the cliched message as follows:

```
@POST
@Consumes("text/plain")
public void postClichedMessage(String message)  {
        // Store the message
}
```

@GET, @PUT, @POST, @DELETE and @HEAD are resource method designator annotations defined by JAX-RS and which correspond to the similarly named HTTP methods.

The followings are parameter based annotatins.
@QueryParam,@FormParam,@PathParam,@MatrixParam,@HeaderParam,@CookieParam

@QueryParam is used to extract query parameters from the Query component of the request URL.
The **@PathParam** and the other parameter-based annotations, @MatrixParam, @HeaderParam, @CookieParam,
@FormParam obey the same rules as @QueryParam.
@MatrixParam extracts information from URL path segments.
@HeaderParam extracts information from the HTTP headers.
@CookieParam extracts information from the cookies declared in cookie related HTTP headers.
@FormParam is slightly special because it extracts information from a request representation that is of the MIME media type "application/x-www-form-urlencoded" and conforms to the encoding specified by HTML forms, as described here. This parameter is very useful for extracting information that is POSTed by HTML forms.

Let us see a complete example on the development of RESTful web service.

```
package com.ddlab.rnd.rest.jersy;
import java.util.Enumeration;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import com.sun.jersey.spi.resource.Singleton;

@Singleton
@Path("/All")
public class RESTFulServiceMethods
{
        @Context
        HttpServletRequest request;

        @Context
        ServletConfig servletConfig;

        @GET @Path("/getValue")
        @Produces("text/plain")
        public String getValue()
        {
                System.out.println(" Inside the methods getValue() using jersey");
                Enumeration paramNames = request.getParameterNames();
                while(paramNames.hasMoreElements())
                {
                        String parName = (String) paramNames.nextElement();
                        System.out.println(parName+"<->"+request.getParameter(parName));
                }
                return "Hello, Welcome to the world of RESTfull Webservices";
        }
```

```
	@GET @Path("/getValueWithPathParam/{name}/{value}")
	@Produces("text/plain")
	public String getValueWithPathParam(@PathParam("name")String name,@PathParam("value")String value)
	{
		System.out.println(" Inside the methods sayHelloWithPathParam() using jersey");
		System.out.println("name----->"+name);
		System.out.println("value----->"+value);
		Enumeration paramNames = request.getParameterNames();
		while(paramNames.hasMoreElements())
		{
			String parName = (String) paramNames.nextElement();
			System.out.println(parName+"<->"+request.getParameter(parName));
		}
		return "Hello, Welcome to the world of RESTfull Webservices "+name+"<===>"+value;
	}

	@POST @Path("/getValueWithFormParam")
	@Produces("text/plain")
	public String getValueWithFormParam(@FormParam("name")String name,@FormParam("value")String value)
	{
		System.out.println(" Inside the methods sayHello() using jersey");
		System.out.println("name----->"+name);
		System.out.println("value----->"+value);
		Enumeration paramNames = request.getParameterNames();
		while(paramNames.hasMoreElements())
		{
			String parName = (String) paramNames.nextElement();
			System.out.println(parName+"<->"+request.getParameter(parName));
		}
		return "Hello, Welcome to the world of RESTfull Webservices "+name+"<===>"+value;
	}

	@GET @Path("/getValueUsingQuery")
	@Produces("text/plain")
	public String getValueUsingQuery( @QueryParam("name")String name , @QueryParam("value")String value)
	{
		System.out.println("Inside Method getSomeResponseValue()");
		System.out.println("name----->"+name);
		System.out.println("value----->"+value);
		Enumeration paramNames = request.getParameterNames();
		while(paramNames.hasMoreElements())
		{
			String parName = (String) paramNames.nextElement();
			System.out.println(parName+"<->"+request.getParameter(parName));
		}
		return "Request parameters are "+name+":"+value;
	}
}
```

```
package com.ddlab.rnd.rest.jersy.util;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;

@Path("/Account")
public class AccountInfo
{
	@Context
	HttpServletRequest request;

	@Context
	ServletConfig servletConfig;

	@POST @Path("/getAccountInfo")
	@Produces("text/plain")
	public String getAccountInfo( @FormParam(value = "accountNo") String accountNo )
	{
		return "You Account Info->"+accountNo;
	}
}
```

The above two classes AccountInfo and RESTFulServiceMethods are two different web service classes and have been defined in two different packages.

The following is the deployment descriptor "web.xml" file for the RESTful application.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>com.ddlab.rnd.rest.jersy</param-value>
    </init-param>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>com.ddlab.rnd.rest.jersy.util</param-value>
    </init-param>
     <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
</web-app>
```

Finally refer to the below build.xml for building the application.

```
<project name="restfulservicejersey" default="createwar" basedir=".">

	<property name="server.src.dir"		value="${basedir}/serversrc" />
	<property name="bin.dir"		value="${basedir}/bin" />
	<property name="lib.dir"		value="${basedir}/lib" />
	<property name="dist.dir"		value="${basedir}/dist" />
	<property name="server.config.dir"	value="${basedir}/serverconfig" />
	<property name="war.dir"		value="${basedir}/${ant.project.name}.war" />
	<property name="webinf.dir"		value="${war.dir}/WEB-INF" />
	<property name="webinf.classes.dir"	value="${war.dir}/WEB-INF/classes" />
	<property name="webinf.lib.dir"		value="${war.dir}/WEB-INF/lib" />

	<!-- Setting the class path -->
	<path id="classpath">
		<fileset dir="${lib.dir}">
			<include name="**/*.jar" />
		</fileset>
	</path>

	<!-- Cleaning operation to delete the files and folders -->
	<target name="clean">
		<delete dir="${war.dir}" />
		<delete dir="${dist.dir}" />
		<delete dir="${bin.dir}" />
	</target>

	<!-- Initialization process to create the necessary directories -->
	<target name="init" depends="clean">
		<mkdir dir="${bin.dir}" />
		<mkdir dir="${dist.dir}" />
		<mkdir dir="${war.dir}" />
		<mkdir dir="${webinf.classes.dir}" />
		<mkdir dir="${webinf.lib.dir}" />
	</target>

	<!-- Compiling the server side source and creation of jar file -->
	<target name="compile" depends="init">
		<echo message="classpath" />
		<javac srcdir="${server.src.dir}" destdir="${bin.dir}" classpathref="classpath" />
		<jar destfile="${dist.dir}/${ant.project.name}.jar" basedir="${bin.dir}" />
		<copy todir="${webinf.lib.dir}" flatten="true">
			<fileset dir="${lib.dir}" includes="**/*.jar" excludes="**/servlet-api.jar" />
			<fileset dir="${dist.dir}">
				<include name="**/*.jar" />
			</fileset>
		</copy>
	</target>

	<!-- Creation of war file -->
	<target name="createwar" depends="compile">
		<copy todir="${webinf.dir}" flatten="true">
			<fileset dir="${server.config.dir}" includes="**/*.xml" />
		</copy>
		<jar destfile="${dist.dir}/${ant.project.name}.war" basedir="${war.dir}" />
	</target>
</project>
```

Libraries required
The following jar files are required to develop the above application.

jar files from Jersey
jersey/asm-3.1.jar
jackson-core-asl-1.5.5.jar
jackson-jaxrs-1.5.5.jar
jackson-mapper-asl-1.5.5.jar
jackson-xc-1.5.5.jar
jersey-client-1.4.jar
jersey-core-1.4.jar
jersey-json-1.4.jar
jersey-server-1.4.jar
jettison-1.1.jar
jsr311-api-1.1.1.jar

jar file from Servlet
servlet-api.jar

How to test
After deploying the web application, you can invoke all the web service methods and test it. To get the WADL file, use the following URL. **http://localhost:8080/restfulservicejersey/application.wadl**.
To check whether you have deployed the correct RESTful web service application, you can access the WADL.

The following is a test class to test all the defined methods for the RESTful web service .

```
import java.net.URI;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.UriBuilder;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import com.sun.jersey.api.representation.Form;
import com.sun.jersey.core.util.MultivaluedMapImpl;

public class TestRestServices {
	private static URI getBaseURI( String endPointUrl ) {
		return UriBuilder.fromUri(endPointUrl).build();
	}

	public static void invokeGetValue()
	{
		//Invocation of the Method /getValue
		String endPointUrl = "http://localhost:8080/restfulservicejersey/All";
		String methodName = "getValue";//Method to invoke
		ClientConfig config = new DefaultClientConfig();
		Client client = Client.create(config);
//		WebResource service = client.resource(getBaseURI(endPointUrl));//You can also use like this
		WebResource service = client.resource(endPointUrl);//You can also use like this

		String responseStr = service.path(methodName).accept(MediaType.TEXT_PLAIN).get(String.class);
		System.out.println("Actual Response from Server : "+responseStr);
		ClientResponse response = service.path("getValue").accept(MediaType.TEXT_PLAIN).get(ClientResponse.class);
		System.out.println("Response Client: "+response.getClient());
		//To know the String response
		String output = response.getEntity(String.class);
		System.out.println("Output from Server .... \n"+output);
	}
```

```
public static void invokeGetValueWithPathParam( String name , String value )
{
	//http://localhost:8080/restfulservicejersey/All/getValueWithPathParam/deba/hati
	/*
	 * You can directly invoke the above URL in the browser to get the contents
	 * or
	 * you can execute the following program to get the result from the server
	 */
	String endPointUrl = "http://localhost:8080/restfulservicejersey/All/getValueWithPathParam/"+name+"/"+value;
	Client client = Client.create();
	WebResource service = client.resource(endPointUrl);
	String responseStr = service.accept(MediaType.TEXT_PLAIN).get(String.class);
	System.out.println("Actual Response from Server : "+responseStr);
}

public static void invokeGetValueWithFormParam( String name , String value )
{
	String endPointUrl = "http://localhost:8080/restfulservicejersey/All/getValueWithFormParam";
	Client client = Client.create();
	WebResource service = client.resource(endPointUrl);

	Form params = new Form();
   params.add("name", name);
   params.add("value", value);

   String response = service.post(String.class,params);
   System.out.println("Response From Server : "+response);
}

public static void invokeGetValueUsingQuery( String name , String value )
{
	String endPointUrl = "http://localhost:8080/restfulservicejersey/All/getValueUsingQuery";
	Client client = Client.create();
	WebResource service = client.resource(endPointUrl);

	MultivaluedMap<String, String> params = new MultivaluedMapImpl();
   params.add("name", "Deba");
   params.add("value", "Y");

   String response = service.queryParams(params).get(String.class);
   System.out.println(response);
}

public static void invokeGetAccountInfo( String accountNo )
{
	String endPointUrl = "http://localhost:8080/restfulservicejersey/Account/getAccountInfo";
	Client client = Client.create();
	WebResource service = client.resource(endPointUrl);

	Form params = new Form();
   params.add("accountNo", accountNo);

   String response = service.post(String.class,params);
   System.out.println("Response From Server : "+response);
}
```

```
public static void main(String[] args)
        {
                invokeGetValue();
                System.out.println("--------- End of getValue() ---------");
                invokeGetValueWithPathParam("Debadatta", "Mishra");
                System.out.println("--------- End of getValueWithPathParam() ---------");
                invokeGetValueWithFormParam("PIKU", "SIR");
                System.out.println("--------- End of getValueWithFormParam() ---------");
                invokeGetValueUsingQuery("ABC", "PQRS");
                System.out.println("--------- End of getValueUsingQuery() ---------");
                invokeGetAccountInfo("A0123456");
        }

}
```

For the above RESTful application, find below the contents of the file "application.wadl".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
   <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Jersey: 1.4 09/11/2010 10:30 PM"/>
   <resources base="http://localhost:8080/restfulservicejersey/">
      <resource path="/Account">
         <resource path="/getAccountInfo">
            <method name="POST" id="getAccountInfo">
               <request>
                  <representation mediaType="application/x-www-form-urlencoded">
                     <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="accountNo"/>
                  </representation>
               </request>
               <response>
                  <representation mediaType="text/plain"/>
               </response>
            </method>
         </resource>
      </resource>
      <resource path="/All">
         <resource path="/getValue">
            <method name="GET" id="getValue">
               <response>
                  <representation mediaType="text/plain"/>
               </response>
            </method>
         </resource>
         <resource path="/getValueUsingQuery">
            <method name="GET" id="getValueUsingQuery">
               <request>
                  <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="name"/>
                  <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="value"/>
               </request>
               <response>
                  <representation mediaType="text/plain"/>
               </response>
            </method>
         </resource>
         <resource path="/getValueWithPathParam/{name}/{value}">
            <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="template" name="name"/>
            <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="template" name="value"/>
            <method name="GET" id="getValueWithPathParam">
               <response>
                  <representation mediaType="text/plain"/>
               </response>
```

```
        </method>
      </resource>
      <resource path="/getValueWithFormParam">
        <method name="POST" id="getValueWithFormParam">
          <request>
            <representation mediaType="application/x-www-form-urlencoded">
              <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="name"/>
              <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="value"/>
            </representation>
          </request>
          <response>
            <representation mediaType="text/plain"/>
          </response>
        </method>
      </resource>
    </resource>
  </resources>
</application>
```

How will you invoke or consume the RESTful web service methods from WADL url ?

Just like how we consume a WSDl file, we can also do the same from an WADL file. There is an API provided by "wadl" from java.net which is used to convert wadl to java.

Capsule :

1. Download the wadl from the following site.
http://search.maven.org/remotecontent?filepath=org/jvnet/ws/wadl/wadl-dist/1.1.1/wadl-dist-1.1.1-bin.zip
2. Put all the relevent jar files in the lib directory.
3. Run the build script to convert from WADL to JAVA. For this we use WJC compiler to convert from wadl to java classes.
4. Finally create the jar file and put it in the classpath.
5. Using the generated jar file, write a program to execute the test cases.

Let us see the complete example below.
Refer to the above application.wadl file.

Let us see the build.xml file which is used to generate the wadl to java.

```
<project name="wadlclientgeneration" default="createJar" basedir=".">
        <property name="bin.dir"                value="${basedir}/bin" />
        <property name="lib.dir"                value="${basedir}/lib" />
        <property name="dist.dir"               value="${basedir}/dist" />
        <property name="build.dir"              value="${basedir}/build" />
        <property name="wadl.src.dir"           value="${basedir}/wadlsrc" />
        <property name="gen.src.dir"            value="${build.dir}/gen-src"/>
        <property name="package.name"           value="com.ddlab.wadl.client"/>
        <property name="jar.file.name"          value="${dist.dir}/wadlclient.jar"/>
        <property name="wadl.name"              value="${wadl.src.dir}/application.wadl"/>

          <!-- Setting the class path -->
          <path id="classpath">
            <fileset dir="${lib.dir}">
              <include name="**/*.jar" />
            </fileset>
          </path>

        <target name="clean">
                <delete dir="${build.dir}" />
                <delete dir="${dist.dir}" />
                <delete dir="${bin.dir}" />
        </target>
```

```
        <!-- Initialization process to create the necessary directories -->
        <target name="init" depends="clean">
                <mkdir dir="${bin.dir}" />
                <mkdir dir="${dist.dir}" />
                <mkdir dir="${build.dir}" />
                <mkdir dir="${build.dir}/gen-src"/>
        </target>

  <taskdef name="wjc" classname="org.jvnet.ws.wadl2java.WJCTask">
    <classpath>
      <fileset dir="${lib.dir}" includes="**/*.jar"
            excludes="wadl-cmdline*.jar"/>
    </classpath>
  </taskdef>

  <target name="generate-sources" depends="init">
    <echo message="Compiling the description..." />
        <echo message="${basedir}"/>
        <echo message="${wadl.src}"/>
        <!-- You have to mention the path of wadl file like absolute path, do not give like Ant dir style -->
    <wjc description="wadlsrc/application.wadl"
        package="${package.name}"
        autoSchemaPackage="true"
        target="${gen.src.dir}">
       <!-- <customizations dir="${basedir}/../share" includes="binding.xjb" /> -->
       <produces dir="${gen.src.dir}"
             includes="**/*.java" />
       <!-- <depends dir="${basedir}/../share"  includes="*.xsd"/> -->
       <depends dir="." includes="build.xml"/>
    </wjc>
  </target>

        <target name="createJar" depends="generate-sources">
                <javac srcdir="${gen.src.dir}" destdir="${bin.dir}" classpathref="classpath"></javac>
                <jar destfile="${jar.file.name}" basedir="${bin.dir}"></jar>
        </target>

</project>
```

NOTE:
While mentioning the path of the wadl file, mention the absolute path of the wadl file, otherwise it will throw Exception. That is why we have mentioned the path of wadl file as "**wadlsrc/application.wadl**" in the description of the WJC task.

The above script will generate the java source file and it will compile and create a jar file.
Put the jar file in the class and run the below java code to execute it.

```
import com.ddlab.wadl.client.Localhost_Restfulservicejersey;
import com.sun.jersey.api.representation.Form;

public class Test
{
	public static void main(String[] args)
	{
		String getValueResponse = Localhost_Restfulservicejersey.all().getValue().getAsTextPlain(String.class);
		System.out.println("Response from Server getValueResponse : "+getValueResponse);

		Form params = new Form();
	   params.add("name", "DDDD");
	   params.add("value", "TTTT");
		String getValueWithFormParamRes = Localhost_Restfulservicejersey
				.all()
				.getValueWithFormParam()
				.postApplicationXWwwFormUrlencodedAsTextPlain(params,
						String.class);
		System.out.println("Response from Server for getValueWithFormParam: "+getValueWithFormParamRes);

		String getValueUsingQueryRes = Localhost_Restfulservicejersey.all()
				.getValueUsingQuery()
				.getAsTextPlain("My Name", "Example", String.class);
		System.out.println("Response from Server for getValueUsingQuery : "+getValueUsingQueryRes);

		String getValueWithPathParamRes = Localhost_Restfulservicejersey.all()
				.getValueWithPathParamNameValue("path1", "value1")
				.getAsTextPlain(String.class);
		System.out.println("Response from Server for getValueWithPathParam: "+getValueWithPathParamRes);

		Form infoParams = new Form();
		infoParams.add("accountNo", "123456789");
		String acctInfoRes =
Localhost_Restfulservicejersey.account().getAccountInfo().postApplicationXWwwFormUrlencodedAsTextPlain(infoParams,
String.class);

		System.out.println("Response from Server for Account Info : "+acctInfoRes);

	}

}
```

REST vs SOAP

There are currently two schools of thought in developing web services:
the traditional, standards-based approach (SOAP) and conceptually
simpler and the trendier new kid on the block (REST).
SOAP has become the standard for exchanging XML-based messages.
The basic structure of SOAP is

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
    <!-- Header information here -->
 </env:Header>
 <env:Body>
   <!-- Body or "Payload" here, a Fault if error happened -->
 </env:Body>
</env:Envelope>
```

The SOAP body contains the "payload" of the message, which is defined by the WSDL's <Message> part. If there is an error that needs to be transmitted back to the sender, a single <Fault> element is used as a child of the <Body>.

REST is an architectural style that can be summed up as four verbs (GET, POST, PUT, and DELETE from HTTP 1.1) . You can get the contents of that object using an HTTP GET; you then might use a POST, PUT, or DELETE to modify the object (in practice most of the services use a POST for this).

Pros and Cons of SOAP

Pros:

Langauge, platform, and transport agnostic
Designed to handle distributed computing environments
Is the prevailing standard for web services, and hence has better support from other standards (WSDL, WS-*) and tooling from vendors
Built-in error handling (faults)
Extensibility

Cons:

Conceptually more difficult, more "heavy-weight" than REST
More verbose
Harder to develop, requires tools

Pros and Cons of REST

Pros:

Language and platform agnostic
Much simpler to develop than SOAP
Small learning curve, less reliance on tools
Concise, no need for additional messaging layer
Closer in design and philosophy to the Web
Bandwidth Usage – REST is Lighter

Cons:

Assumes a point-to-point communication model--not usable for distributed computing environment where message may go through one or more intermediaries
Lack of standards support for security, policy, reliable messaging, etc., so services that have more sophisticated requirements are harder to develop ("roll your own")
Tied to the HTTP transport model

The basic structure of WADL file is given below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
    <doc/>

    <resources base="">
        <resource path="">
            <resource path="">
                <method name="POST" id="getAccountInfo">
                    <request>
                        <representation mediaType="">
                            <param xmlns:xs="" type="xs:string" style="query" name="accountNo"/>
                        </representation>
                    </request>
                    <response>
                        <representation mediaType="text/plain"/>
                    </response>
                </method>
            </resource>
        </resource>
    </resources>
</application>
```

A complete example is given below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
    <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Jersey: 1.4 09/11/2010
10:30 PM"/>
    <resources base="http://localhost:8080/restfulservicejersey/">
        <resource path="/Account">
            <resource path="/getAccountInfo">
                <method name="POST" id="getAccountInfo">
                    <request>
                        <representation mediaType="application/x-www-form-urlencoded">
                            <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
style="query" name="accountNo"/>
                        </representation>
                    </request>
                    <response>
                        <representation mediaType="text/plain"/>
                    </response>
                </method>
            </resource>
        </resource>
    </resources>
</application>
```

HATEOAS

From Wikipedia, the free encyclopedia

HATEOAS, an abbreviation for **Hypermedia as the Engine of Application State**, is a constraint of the REST application architecture that distinguishes it from most other network application architectures. The principle is that a client interacts with a network application entirely through hypermedia provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia. By contrast, in some service-oriented architectures (SOA), clients and servers interact through a fixed interface shared through documentation or an interface description language (IDL).

The HATEOAS constraint decouples client and server in a way that allows the server functionality to evolve independently.

Details[edit]

A REST client enters a REST application through a simple fixed URL. All future actions the client may take are discovered within resource representations returned from the server. Themedia types used for these representations, and the link relations they may contain, are standardized. The client transitions through application states by selecting from the links within a representation or by manipulating the representation in other ways afforded by its media type. In this way, RESTful interaction is driven by hypermedia, rather than out-of-band information.[1]

For example [2] here is a GET request to fetch an Account resource, requesting details in an XML representation:

```
GET /account/12345 HTTP/1.1
Host: somebank.org
Accept: application/xml
...
```

Here is the response:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
   <account_number>12345</account_number>
   <balance currency="usd">100.00</balance>
   <link rel="deposit" href="http://somebank.org/account/12345/deposit" />
   <link rel="withdraw" href="http://somebank.org/account/12345/withdraw" />
   <link rel="transfer" href="http://somebank.org/account/12345/transfer" />
   <link rel="close" href="http://somebank.org/account/12345/close" />
 </account>
```

Note the response contains 4 possible follow-up links - to make a deposit, a withdrawal, a transfer or to close the account.

Some time later the account information is retrieved again, but now the account is overdrawn:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">-25.00</balance>
    <link rel="deposit" href="http://somebank.org/account/12345/deposit" />
</account>
```

Now only one link is available: to deposit more money. In its current *state*, the other links are not available. Hence the term *Engine of Application State*. What actions are possible vary as the state of the resource varies.

A client does not need to understand every media type and communication mechanism offered by the server. The ability to understand new media types can be acquired at run-time through "code-on-demand" provided to the client by the server.[3]

Origins[edit]

The HATEOAS constraint is an essential part of the "uniform interface" feature of REST, as defined in Roy Fielding's doctoral dissertation.[3] Fielding has further described the concept on his blog.[1]

The purpose of some of the strictness of this and other REST constraints, Fielding explains, is "software design on the scale of decades: every detail is intended to promote software longevity and independent evolution. Many of the constraints are directly opposed to short-term efficiency. Unfortunately, people are fairly good at short-term design, and usually awful at long-term design".[1]

https://spring.io/understanding/HATEOAS

Understanding HATEOAS

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture.

A hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses. This capability differs from that of SOA-based systems and WSDL-driven interfaces. With SOA, servers and clients usually must access a fixed specification that might be staged somewhere else on the website, on another website, or perhaps distributed by email.

Note: Pronunciations of HATEOAS vary. Some people pronounce it as "hate-ee-os," similar to "hideous," or as "hate O-A-S". People also refer to it as a hypermedia-driven system.

Examples

The following code represents a `Customer` object.

```
class Customer {

    String name;

}
```

A simple JSON presentation is traditionally rendered as:

```
{

    "name" : "Alice"

}
```

The customer data is there, but the data contains nothing about its relevant links.

A HATEOAS-based response would look like this:

```
{

    "name": "Alice",

    "links": [ {

        "rel": "self",

        "href": "http://localhost:8080/customer/1"

    } ]
```

```
}
```

This response not only has the person's name, but includes the self-linking URL where that person is located.

- **rel** means relationship. In this case, it's a self-referencing hyperlink. More complex systems might include other relationships. For example, an order might have a `"rel":"customer"` relationship, linking the order to its customer.
- **href** is a complete URL that uniquely defines the resource.

Note: Although these responses are shown in JSON, XML is also accepted as a standard response format. HATEOAS doesn't impose the requirement of either format. The hypermedia links are the focus of HATEOAS.

It's possible to build more complex relationships. With HATEOAS, the output makes it easy to glean how to interact with the service without looking up a specification or other external document.

Look at the following catalog, courtesy of the sample application shown in the Spring Data Book:

```
{

    "content": [ {

        "price": 499.00,

        "description": "Apple tablet device",

        "name": "iPad",

        "links": [ {

            "rel": "self",

            "href": "http://localhost:8080/product/1"

        } ],

        "attributes": {

            "connector": "socket"

        }

    }, {

        "price": 49.00,

        "description": "Dock for iPhone/iPad",

        "name": "Dock",

        "links": [ {
```

```
            "rel": "self",
            "href": "http://localhost:8080/product/3"
        } ],
        "attributes": {
            "connector": "plug"
        }
    } ],
    "links": [ {
        "rel": "product.search",
        "href": "http://localhost:8080/product/search"
    } ]
}
```

Not only are the items and their prices shown, but the URL for each resource is shown, providing clear information on how to access these resources. According to the Richardson Maturity Model, HATEOAS is considered the final level of REST. This means that each link is presumed to implement the standard REST verbs of GET, POST, PUT, and DELETE (or a subset). Thus providing the links as shown above gives the client the information they need to navigate the service.

What HATEOAS actually means

RESTless about RESTful

These days there is much discussion about REST and HATEOAS, and many people feel urged to reinterpret what HATEOAS means or what Roy Fielding's often-cited dissertation allegedly would say in *their* understanding, and what HATEOAS should be implemented like therefore. While I first felt amused about this "dispute about nothing" (just ask Mr Fielding if you don't understand what his dissertation tells us; no need to *guess*), the longer I follow those (in part ridiculously wrong)*assumptions* and *myths*, the more *myself* feels urged to stop those and shout: "Guys, *before* discussing *your* ideas, first learn what *Mr Fielding's* idea was!" There is nothing to *interprete* or *construe*. *His* words are clear and *unambiguous*. Just read them, if necessary twice. It tells us everything we like to know. Really.

FLASHBACK

In his dissertation Roy Thomas Fielding explained RESTful architecture (actually it seems that it even *introduced* the word REST), including hypermedia as the engine of application state (HATEOAS):

*"The next control state of an application resides **in the representation of the** first **requested resource**, … The application state is controlled and stored **by the user agent** … anticipate changes to that state (e.g., **link maps** and prefetching of representations) … The model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions **in the current set of representations**."*

Okay, so what does that mean and why is most of that wrong what currently is discussed as proposed implementations of HATEOAS?

To understand Fielding's above explanation, we have to remember what his dissertation was about. Fielding was a contributor to the HTTP standard. In his research he discovered that the method of operation of the world wide web can be abstracted to a general architecture he called REpresentational State Transfer (REST). The thesis behind REST is: Since the WWW is working and scaling so perfectly, while REST is the WWW's architecture, REST itself will be working and scaling well in other domains too, possibly outside of the WWW. In fact he is true, which is why we all are so crazy about REST these days. In detail he identified four key factors that REST is comprising:

*"REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and,**hypermedia as the engine of application state**."*

Speaking in techniques of the WWW (which is the implementation mostly used to apply the abstract idea of REST to real, physical applications), those four core ideas actually would be the combination of: Using **URI**s to transfer **MIME-typed documents** with **GET / PUT / POST / DELETE** (like their counterparts SELECT / UPDATE / INSERT / DELETE would apply the same idea to SQL) and …? It is the last ellipsis this blog entry (and HATEOAS) is about.

```
WWW := URI + HTTP + MIME + Hypermedia
```

WHAT ARE HATEOAS AND HYPERMEDIA?

HATEOAS is the short form of "*hypermedia* as the engine of state", as we learn from the disseration. But what does it mean? Let's start with "state". "State" means the current status of the sum of information found in the system at a particular point in time. For example, if we have an order, it will have (at least) two states: Either it is sent to the customer, or it is not (certainly "state" is neither restricted to a single variable nor a particular type like boolean; typically state is a complex construct of several informations). So what is an "engine of state"? As the example shows, most objects typically have not statically one state for an infinite time, but will change its state from time to time. An order was not sent to the customer, then got sent, so its new state now is "sent" now. It *transitioned* it's state due to an *action*. The move from one state to another is called "state transition" and the part of the system that controls the state transitions (i. e. applies a rule set defining what action will result in which state transition, e. g. "if current state is 'new' and action is 'wear' then new state is 'used'") is called a *state engine*.

So now that we know what a state engine is, let's look at hypermedia:

***Hypermedia** is used as a logical extension of the term hypertext in which graphics, audio, video, plain text and hyperlinks intertwine to create a generally non-linear medium of information.* (from WIKIPEDIA)

Or in clear words: If two or more documents are related *by a link*, this media is called hypermedia. But this is not what hypermedia in the inner sense of the WWW means, so let's once more cite Mr Fielding's dissertation:

*"The model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions **in** the current set of representations."*

The bold "**in**" is essential to understand what actually *is* HATEOAS and what is *not*: *Only* if the alternative state transitions are found **in** the representations (i. e. **in**the MIME-typed documents that actually render state, e. g. XML, HTML or PDF documents, audio files or video streams) -but *not* aside or just near of them- then HATEOAS is true. Why? Because exactly that is what the word HATEOAS itself tells:

```
HATEOAS := Hypermedia as the engine of state
```

Hypermedia as the state of transtition. It is neither "*the transport protocol* as the state of transition", nor is it "*something beside the representation* as the state of transation". It is clearly *hypermedia* and nothing else and even more clearly it is exactly *the representation*. And the representation is *only* the MIME-typed document but not any headers, URIs, parameters, or whatever people currently are

discussing. Unfortunately Mr Fielding used two divided sentences to explain the concept. It would be much clearer and free of discussion if he would have written what he actually meant to say:

```
HATEOAS := hypermedia documents as the state of state
```

Why didn't he do so? Let's again check his background: He analyzed the WWW, which comprises mostly of HTML files. HTML files are hypermedia documents. They*contain* links (relations) to other documents by elements. I really can understand that he never would have imagined that anybody would ever have the really strange idea to cut away the links out from the hypermedia documents and store them elsewhere and still call that document hypermedia. It would break the whole idea of the WWW if you would remove all elements from all HTML files and store them somewhere else. But that is exactly what people currently are discussing (not for HTML but for different formats)! Moreover, I suspect that he was just used to call even one single HTML file hypermedia due to its theoretical possibility to point to a second one by an element. Or in other words: Fielding's "hypermedia" in fact is *not* any different part of the system but *solely* the document. This is why he wrote in the above cite explicitly that the state transitions are found **in** the representations. It was just absolutely clear that it makes no sense to have the links outside, as it is not common in the WWW.

Update: BTW, yes, it is RESTful to put links in the "Link" header when using HTTP, as Mr Fielding told me. But don't expect any REST client to take care of that, least of them will, unless there is a good reason (like the entity being an image), just as a browser ignores any s in HTML unless it is a relation it *likes* to handle (like RSS feeds). So it is valid, but of potential risk to do so.

What to learn from Mr Fielding?

There is only and exactly one valid implementation of HATEOAS: Having **links inside of the document** (or, *with care*, in the "Link" HTTP header, if HTTP is used).

Just like HTML files link to each other with tags, or just like XML files link to each other with XLink, *all* state transfer has to be done *solely* as a reaction to a link found *inside* of a representation (i. e. inside of a MIME-typed document). Any other technology, like passing the state, possible actions or links outside of the representation, e. g. in HTTP headers etc., is *by definition* **not** HATEOAS. Moreover, the weird idea of having explicit URIs solely for state transition without passing any document in or getting any document back, is **not** HATEOAS. Looking once more at the concept of the WWW, there typically are no such "pure action links". The typical state transfer in HTML is done by either passing a modified version of the document, or by passing a form containing information to be merged into the new state. But *never* will it be HATEOAS to invoke a link without passing new state. Why?
Once more, the dissertation is providing an absolutely clear an unambiguous definition:
"The application state is controlled and stored by the user agent … freeing the server from the scalability problems of storing state …"
When you are trying to execute a typical "solution" currently widely discussed

```
POST http://resource/id/the-action-to-execute
```

then this will ask *the server* to do the state transition. **This is *in diametral contrast* to the above cite of the dissertation** which clearly says that it is *not the server but solely the client* that stores and controls state and **thus is explicitly *not* HATEOAS**. It just makes no sense to call a special URI to trigger *a server side* action if*the client* already has switched to the new state. And it shall be *the client* that does the switch but *not the server*. You can just call the "normal" URI of the resource and pass the already state-transisioned version of the resource. By doing so, the server will implicitly learn the new state. No need to tell the server which action was responsible for that (if that would be needed from a business aspect, then it *must* be part of the uploaded document but not of the used transfer protocol)!

SO HOW TO DO HATEOAS THE RIGHT WAY?

In short: Let your client read the possible actions as URIs out of the received document, set up the modified document and then invoke those link. No clear yet? Let's make an example. We have an order that is not shipped, and now we want to issue shipping. So the client GETs the order (e. g. as an XML document) and inspects its content. Inside it finds XLinks for various actions. One of them is for shipping. The client puts together the needed information from shipping and POSTs that to the XLink's target. In RESTful terms, what we do is creating a new instance of a shipping instruction by uploading an XML containing shipping details (typically containing the order document itself as a child element, or more simple, its URI as a reference). How does our client know what of the contained XLink URIs the one for shipping is? This is a case of definition. XLink for example comes with the role attribute, so we could defined that it must be the one with the role set to "ship".
It's just similar in case your client is a WWW browser and your document format is HTML: You download the order as a HTML file, containing links. You click on the button that has the title "ship" which performs an action of PUT, containing the shipping details you filled in manually. How did you now which button you must press? You don't. You just guessed well or you had clear instructions.

So to come back to XML, it is just a question of clear instructions, which means, definitions: Your client just needs to know that the XLink to search for is called "ship". There is no technical solution. At least this piece of information must exist. If man does not know that the english word for sending something away is "ship", he wouldn't find the button, too.

And other media types? Well, what will man do when receiving a media file that he has no player for? Nothing! Same for machines. The client needs to be aware of the used media types. It is impossible for the client machine to deal with unknown media types, just as it is impossible for the browser. A good idea would be to use an abstract API that allows to plug in media handlers, just as browsers do.

http://www.allapplabs.com/glossary/idempotent.htm

Idempotent

If methods are written in such a way that repeated calls to the same method do not cause duplicate updates, the method is said to be "idempotent."

In mathematics an idempotent element, or an idempotent for short, is anything that, when multiplied by itself, gives itself as result. For example, the only two real numbers which are idempotent are 0 and 1.

In user interface design, a button can be called "idempotent" if pressing it more than once will have the same effect as pressing it once. For example, a "Pause" button is not idempotent if it toggles the paused state. On the other hand, if pressing it multiple times keeps the system paused and pressing "Play" resumes, then "Pause" is idempotent. This is useful in interfaces such as infrared remote controls and touch screens where the user may not be sure of having pressed the button successfully and may press it again. Elevator call buttons are also idempotent, though many people think they are not.

http://www.restapitutorial.com/lessons/idempotency.html

Idempotence

Idempotence is a funky word that often hooks people. Idempotence is sometimes a confusing concept, at least from the academic definition.

From a RESTful service standpoint, for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request. Note that while idempotent operations produce the same result on the server (no side effects), the response itself may not be the same (e.g. a resource's state may change between requests).

The PUT and DELETE methods are defined to be idempotent. However, there is a caveat on DELETE. The problem with DELETE, which if successful would normally return a 200 (OK) or 204 (No Content), will often return a 404 (Not Found) on subsequent calls, unless the service is configured to "mark" resources for deletion without actually deleting them. However, when the service actually deletes the resource, the next call will not find the resource to delete it and return a 404. However, the state on the server is the same after each DELETE call, but the response is different.

GET, HEAD, OPTIONS and TRACE methods are defined as safe, meaning they are only intended for retrieving data. This makes them idempotent as well since multiple, identical requests will behave the same.

http://stackoverflow.com/questions/1077412/what-is-an-idempotent-operation

An idempotent operation can be repeated an arbitrary number of times and the result will be the same as if it had been done only once. In arithmetic, adding zero to a number is idempotent.

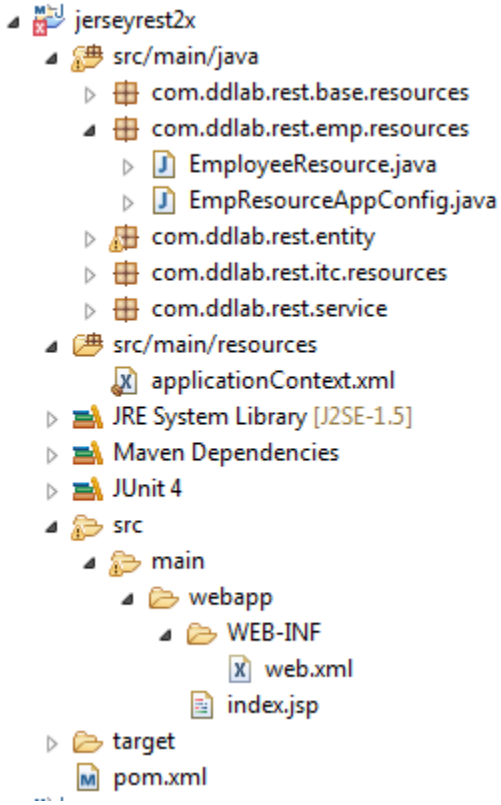
Idempotence is talked about a lot in the context of "RESTful" web services. REST seeks to maximally leverage HTTP to give programs access to web content, and is usually set in contrast to SOAP-based web services, which just tunnel remote procedure call style services inside HTTP requests and responses.

REST organizes a web application into "resources" (like a Twitter user, or a Flickr image) and then uses the HTTP verbs of POST, PUT, GET, and DELETE to create, update, read, and delete those resources.

Idempotence plays an important role in REST. If you GET a representation of a REST resource (eg, GET a jpeg image from Flickr), and the operation fails, you can just repeat the GET again and again until the operation succeeds. To the web service, it doesn't matter how many times the image is gotten. Likewise, if you use a RESTful web service to update your Twitter account information, you can PUT the new information as many times as it takes in order to get confirmation from the web service. PUT-ing it a thousand times is the same as PUT-ing it once. Similarly DELETE-ing a REST resource a thousand times is the same as deleting it once. Idempotence thus makes it a lot easier to construct a web service that's resilient to communication errors.

Further reading: RESTful Web Services, by Richardson and Ruby (idempotence is discussed on page 103-104), and Roy Fielding's PhD dissertation on REST. Fielding was one of the authors of HTTP 1.1, RFC-2616, which talks about idempotence in section 9.1.2.

Project Structure



Maven Config (pom.xml)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>jerseyrest2x</groupId>
	<artifactId>jerseyrest2x</artifactId>
	<packaging>war</packaging>
	<version>0.0.1-SNAPSHOT</version>
	<name>jerseyrest2x Maven Webapp</name>
	<url>http://maven.apache.org</url>
	<dependencies>
		<!-- servlet 3.0 -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>3.0.1</version>
			<scope>provided</scope>
		</dependency>
		<!-- jax-rs 2.0 -->
		<dependency>
			<groupId>javax.ws.rs</groupId>
			<artifactId>javax.ws.rs-api</artifactId>
			<version>2.0</version>
		</dependency>
		<!-- jersey implementation library -->
		<dependency>
			<groupId>org.glassfish.jersey.containers</groupId>
```

```
            <artifactId>jersey-container-servlet</artifactId>
            <!-- <version>2.11</version> -->
            <version>2.14</version>
        </dependency>
        <!-- jersey media multipart for file upload -->
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-multipart</artifactId>
            <!-- <version>2.11</version> -->
            <version>2.14</version>
        </dependency>
        <!-- Spring Jersey Integration -->
        <dependency>
            <groupId>org.glassfish.jersey.ext</groupId>
            <artifactId>jersey-spring3</artifactId>
            <version>2.14</version>
        </dependency>

        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>
        <dependency>
            <groupId>org.codehaus.jackson</groupId>
            <artifactId>jackson-core-asl</artifactId>
            <version>1.9.13</version>
        </dependency>
        <dependency>
            <groupId>org.codehaus.jackson</groupId>
            <artifactId>jackson-mapper-asl</artifactId>
            <version>1.9.13</version>
        </dependency>
        <!-- The following is required for object to json -->
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-json-jackson</artifactId>
            <version>2.14</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>jerseyrest2x</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.0</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.4</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

WEB.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">
      <display-name>JAX-RS 2 File Upload Example Web Application</display-name>
      <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>

</web-app>
```

Spring IOC Configuration (applicationContext.xml)

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.0.xsd">

      <context:component-scan base-package="com.ddlab" />

      <bean id="dbHandler" class="com.ddlab.rest.service.EmployeeServiceImpl" />

</beans>
```

Java Classes

BaseResource.java

```
package com.ddlab.rest.base.resources;
import java.nio.file.AccessDeniedException;
import javax.ws.rs.NotFoundException;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

public class BaseResource {
      protected WebApplicationException createWebappException(Exception incomingException) {

            Status status;
            String string = null;
            if (incomingException instanceof SecurityException || incomingException instanceof
AccessDeniedException) {
                  status = Status.FORBIDDEN;
            } else if (incomingException instanceof IllegalArgumentException) {
                  status = Status.BAD_REQUEST;
                  string=incomingException.getMessage();
            } else if (incomingException instanceof NotFoundException) {
                  status = Status.NOT_FOUND;
            } else if (incomingException instanceof Exception) {
```

```
                status = Status.INTERNAL_SERVER_ERROR;
            } else {
                status = Status.INTERNAL_SERVER_ERROR;
            }
            return new
WebApplicationException(Response.status(status).entity(string).type("text/plain").build());
      }
}
```

EmployeeResource.java

```
package com.ddlab.rest.emp.resources;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.Response.Status;
import org.springframework.beans.factory.annotation.Autowired;
import com.ddlab.rest.base.resources.BaseResource;
import com.ddlab.rest.entity.User;
import com.ddlab.rest.service.EmployeeService;

@Path("1/empresource")//Vesrion/Resources
public class EmployeeResource extends BaseResource {

      @Context
      HttpServletRequest request;

      @Context
      ServletConfig servletConfig;

      @Autowired
      private EmployeeService empService;

      @Path("/create")
      @POST
      @Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      public Response createUser(User user) {
            //DELETE http://localhost:8080/jerseyrest2x/empresources/1/empresource/create
            //Content-Type:application/json
            //Payload : {"username":"Deba","password":"deba","id":"123"}
            ResponseBuilder responseBuilder = null;
            try {
                  if(user.getUserName() == null || user.getPassword() == null )
                        responseBuilder = Response.status(Status.BAD_REQUEST).entity("Incorrect
password");
                  empService.createUser(user);
```

```
                responseBuilder = Response.status(Status.CREATED).entity("User created
successfully");
            } catch (Exception e) {
                responseBuilder = Response.serverError();
            }
            return responseBuilder.build();
      }

      @Path("/update")
      @PUT
      @Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      public void updateUser(User user) { //204 No content
            //DELETE http://localhost:8080/jerseyrest2x/empresources/1/empresource/update
            //Content-Type:application/json
            //Payload : {"username":"Deba","password":"deba","id":"123"}
            if(user.getUserName() == null || user.getPassword() == null )
                  throw createWebappException(new IllegalArgumentException("Incorrect credentials"));
            empService.updateUser(user);
      }

      @Path("/delete")
      @DELETE
      @Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      public String deleteUser(User user) {
            //DELETE http://localhost:8080/jerseyrest2x/empresources/1/empresource/delete
            //Content-Type:application/json
            //Payload : {"username":"Deba","password":"deba","id":"123"}
            if(user.getUserName() == null || user.getPassword() == null )
                  throw createWebappException(new IllegalArgumentException("Incorrect credentials"));
            empService.deleteUser(user);
            return "User "+user.getUserName()+" has been removed from the system successfully";
      }

      @Path("/userid")
      @GET
      @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      public User getUser(@QueryParam("id") int id) {
            //GET http://localhost:8080/jerseyrest2x/empresources/1/empresource/userid?id=123
            //Accept:application/json or application/xml
            if(id == 0 )
                  throw createWebappException(new IllegalArgumentException("Incorrect ID"));
            return empService.getUserById(String.valueOf(id));
      }

      @Path("/show/image")
      @GET
      @Produces("image/png")
      public Response showImage() {
            //GET http://localhost:8080/jerseyrest2x/empresources/1/empresource/show/image
            //You can directly hit in the browser
            InputStream inStream = null;
            try {
                  String SERVER_DOWNLOAD_LOCATION_FOLDER = "D:"+File.separator+"temp"+File.separator;
                  String IMG_FILE_NAME = "r.jpg";
                  inStream = new FileInputStream(SERVER_DOWNLOAD_LOCATION_FOLDER+IMG_FILE_NAME);
            } catch (FileNotFoundException e) {
                  e.printStackTrace();
            }
            return Response.ok().entity(inStream).build();
      }
}
```

EmpResourceAppConfig.java

```
package com.ddlab.rest.emp.resources;
import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
import org.glassfish.jersey.jackson.JacksonFeature;
import org.glassfish.jersey.media.multipart.MultiPartFeature;

//@ApplicationPath("/*")
@ApplicationPath("empresources")
public class EmpResourceAppConfig extends Application {
//There can be multiple Classes which extend javax.ws.rs.core.Application
//This is used for logical functional separation
	@Override
	public Set<Class<?>> getClasses() {
		Set<Class<?>> resources = new HashSet<Class<?>>();
		resources.add(EmployeeResource.class);
		resources.add(MultiPartFeature.class);
		resources.add(JacksonFeature.class);
		return resources;
	}
}
```

ITCInfotechServices.java

```
package com.ddlab.rest.itc.resources;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.glassfish.jersey.media.multipart.ContentDisposition;
import org.glassfish.jersey.media.multipart.FormDataBodyPart;
import org.glassfish.jersey.media.multipart.FormDataContentDisposition;
import org.glassfish.jersey.media.multipart.FormDataMultiPart;
import org.glassfish.jersey.media.multipart.FormDataParam;
import com.ddlab.rest.entity.Constants;
import com.ddlab.rest.entity.Employee;
import com.ddlab.rest.entity.FileUtil;
```

```
@Path("itc")
public class ITCInfotechServices {

	@Context
	HttpServletRequest request;

	@Context
	ServletConfig servletConfig;

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getOrganisationName() {
		//GET http://localhost:8080/jerseyrest2x/itcresources/itc
		return "ITC Infotech, Bangalore, Karnataka";
	}

	@Path("/address")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddress() {
		// URL : http://localhost:8080/jerseyrest2x/itcresources/itc/address
		return "ITC Infotech India Limited, 18, Banaswadi Main Rd, Maruthi Sevanagar, Bangalore,
560005";
	}

	// ~~~~~~~~~~~~~~~~~~~~~~~~ @PathParam ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	@Path("/address/{areaCode}")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddressByCode(@PathParam("areaCode") String areaCode) {
		//		GET http://localhost:8080/jerseyrest2x/itcresources/itc/address/USA
		try {
			if (areaCode.equalsIgnoreCase("USA"))
				return "12 North State, Route 17,Suite 303,Paramus,New Jersey,NJ-07652";
			else if (areaCode.equalsIgnoreCase("Europe"))
				return "Newell Consulting Oy,P.O. Box 16 , Olari,02211 Espoo, Helsinki";
			else if (areaCode.equalsIgnoreCase("Africa"))
				return "Johannesburg,2nd Floor, West Tower,Nelson Mandela Square,Maude Street,
Sandton,Johannesburg, 2196";
			else if (areaCode.equalsIgnoreCase("Asia"))
				return "ITC Infotech India Limited, 18, Banaswadi Main Rd, Maruthi Sevanagar,
Bangalore, 560005";
			else
				return "No such area code exists for ITC";
		} catch (Exception e) {
			return "No such area code exists for ITC";
		}
	}

	// ~~~~~~~~~~~~~~~~~~~~~~~~ @QueryParam ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	@Path("/regionaladdress/{areaCode}")
	// USA,EUROPE,ASIA
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddressByCountry(@PathParam("areaCode") String areaCode,
			@QueryParam("country") String country) {
		// GET
http://localhost:8080/jerseyrest2x/itcresources/itc/regionaladdress/Europe?country=FI
		try {
			if (areaCode.equalsIgnoreCase("USA")
					&& country.equalsIgnoreCase("NJ"))
				return "12 North State, Route 17,Suite 303,Paramus,New Jersey,NJ-07652";
```

```
                else if (areaCode.equalsIgnoreCase("Europe")
                              && country.equalsIgnoreCase("FI"))
                       return "Newell Consulting Oy,P.O. Box 16 , Olari,02211 Espoo, Helsinki";
                else if (areaCode.equalsIgnoreCase("Europe")
                              && country.equalsIgnoreCase("SE"))
                       return "C/o Matrisen AB,Box 22059 , 104 22 Stockholm";
                else if (areaCode.equalsIgnoreCase("Europe")
                              && country.equalsIgnoreCase("DK"))
                       return "Havnegade 39, 3. sal,1058 Copenhagen K";
                else if (areaCode.equalsIgnoreCase("Asia")
                              && country.equalsIgnoreCase("IN"))
                       return "ITC Infotech India Limited, 18, Banaswadi Main Rd, Maruthi Sevanagar,
Bangalore, 560005";
                else
                       return "No such area code exists for ITC";
          } catch (Exception e) {
                return "No such area code exists for ITC";
          }
    }


    @Path("/itcaddress")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getITCAddress(@MatrixParam("country") String country,
                @MatrixParam("areacode") String areaCode) {
          // GET
http://localhost:8080/jerseyrest2x/itcresources/itc/itcaddress;country=FI;areacode=europe
          return getAddressByCountry(areaCode, country);
    }

    // ~~~~~~~~~~~~~~~~~~~~~~~~ @FormParam ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    @Path("/postaddress")
    @POST
    @Produces(MediaType.TEXT_PLAIN)
    public String postNGetITCAddress(@FormParam("country") String country,
                @FormParam("areacode") String areaCode) {
          // http://localhost:8080/jerseyrest2x/itcresources/itc/postaddress
          return getAddressByCountry(areaCode, country);
    }

    /*
     * In case of Firefox Rest Client
          Set in the header
          “name” = “Content-Type” and “value” = “application/x-www-form-urlencoded”
          Then set the header as
          country - FI
          areacode - europe

          In case of Chrome Postman
          Click on x-www-form-urlencoded in Postman
          country - FI
          areacode - europe
     */

    // ~~~~~~~~~~~~~~~~~~~~~~~~ @PathParam with Object ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    @Path("/emp/{id}")
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public Response getEmployee(@PathParam("id") String id) {
          //GET http://localhost:8080/jerseyrest2x/itcresources/itc/emp/123
          ObjectMapper mapper = new ObjectMapper();
          Employee emp = null;
```

```
            try {
                  File file = new File(Constants.DATA_LOCATION_FOLDER+File.separator+id+".json");
                  if( ! file.exists() )
                        return Response.status(404).entity("No information found for the given
employee id ...").build();
                  emp = mapper.readValue(file, Employee.class);
            } catch (JsonParseException e) {
                  e.printStackTrace();
            } catch (JsonMappingException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            return Response.ok(emp).build();
      }

      /**
       * POST http://localhost:8080/jerseyrest2x/itcresources/itc/createemp
       *
       * JSON Structure
       * {
                "Name": "Deba",
                "Age": 23,
                "Email": "deba@gmail.com",
                "Adrs": {
                    "DoorNo": "12-A",
                    "Street": "Street-11",
                    "City": "Bangalore",
                    "Country": "Karnataka"
                }
            }

            XML Structure
                  <Emp>
                      <Name>Deba</Name>
                      <Age>23</Age>
                      <Email>deba@gmail.com</Email>
                      <Adrs>
                          <DoorNo>12-A</DoorNo>
                          <Street>Street-11</Street>
                          <City>Bangalore</City>
                          <Country>Karnataka</Country>
                      </Adrs>
                 </Emp>
       */
      @Path("/createemp")
      @POST
      @Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      public Response createEmp(Employee emp) {
            System.out.println("Employee object received successfully .........");
            //store into file system
            if( new File(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json").exists())
{
                  return Response.status(409).entity("Employee already exists in the system
...").build();
            }
            FileUtil.saveFile(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json",
emp.toJSON());
            return Response.ok("Employee created successfully...").build();
      }

      /**
```

```
	 * PUT http://localhost:8080/jerseyrest2x/itcresources/itc/updateemp
	 *
	 * JSON Structure
	 * {
	 		"Name": "Deba",
	 		"Age": 23,
	 		"Email": "deba@gmail.com",
	 		"Adrs": {
	 		    "DoorNo": "12-A",
	 		    "Street": "Street-11",
	 		    "City": "Bangalore",
	 		    "Country": "Karnataka"
	 		}
	 	}

	 	XML Structure
	 	   <Emp>
	 	       <Name>Deba</Name>
	 	       <Age>23</Age>
	 	       <Email>deba@gmail.com</Email>
	 	       <Adrs>
	 	           <DoorNo>12-A</DoorNo>
	 	           <Street>Street-11</Street>
	 	           <City>Bangalore</City>
	 	           <Country>Karnataka</Country>
	 	       </Adrs>
	 	   </Emp>
	 */
	@Path("/updateemp")
	@PUT
	@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public Response updateEmp(Employee emp) {
		System.out.println("Employee object received successfully .........");
		//store into file system
		if( new File(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json").exists())
{

	FileUtil.saveFile(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json",
emp.toJSON());
			return Response.ok("Employee info updated successfully...").build();

		}
		else
			return Response.status(404).entity("Employee information not found, hence can not be
updated ...").build();
	}

	/**
	 * PUT http://localhost:8080/jerseyrest2x/itcresources/itc/deleteemp
	 *
	 * JSON Structure
	 * {
	 		"Name": "Deba",
	 		"Age": 23,
	 		"Email": "deba@gmail.com",
	 		"Adrs": {
	 		    "DoorNo": "12-A",
	 		    "Street": "Street-11",
	 		    "City": "Bangalore",
	 		    "Country": "Karnataka"
	 		}
	 	}
```

```
		XML Structure
			<Emp>
			    <Name>Deba</Name>
			    <Age>23</Age>
			    <Email>deba@gmail.com</Email>
			    <Adrs>
			        <DoorNo>12-A</DoorNo>
			        <Street>Street-11</Street>
			        <City>Bangalore</City>
			        <Country>Karnataka</Country>
			    </Adrs>
		    </Emp>
	 */
	@Path("/deleteemp")
	@DELETE
	@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public Response deleteEmp(Employee emp) {
		System.out.println("Employee object received successfully .........");
		//store into file system
		File file = new File(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json");
		if( file.exists()) {
			file.delete();
			return Response.ok("Employee info deleted successfully...").build();

		}
		else
			return Response.status(404).entity("Employee information not found, hence can not be
deleted ...").build();
	}

	@POST
	@Path("/upload")
	@Consumes(MediaType.MULTIPART_FORM_DATA)
	public Response uploadFile(FormDataMultiPart form) {
		//POST http://localhost:8080/jerseyrest2x/itcresources/itc/upload
		//Content-Type:multipart/form-data
		FormDataBodyPart filePart = form.getField("file");
		ContentDisposition headerOfFilePart = filePart.getContentDisposition();
		InputStream fileInputStream = filePart.getValueAs(InputStream.class);
		String filePath = Constants.SERVER_UPLOAD_LOCATION_FOLDER
				+ headerOfFilePart.getFileName();
		// save the file to the server
		saveFile(fileInputStream, filePath);
		String output = "File saved to server location using FormDataMultiPart : "
				+ filePath;
		try {
			fileInputStream.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return Response.status(200).entity(output).build();
	}

	@POST
	@Path("/upload1")
	@Consumes(MediaType.MULTIPART_FORM_DATA)
	public Response uploadFile(
			@FormDataParam("file") InputStream fileInputStream,
			@FormDataParam("file") FormDataContentDisposition contentDispositionHeader) {
		//POST http://localhost:8080/jerseyrest2x/itcresources/itc/upload1
		//Content-Type:multipart/form-data
```

```
		String filePath = Constants.SERVER_UPLOAD_LOCATION_FOLDER +
contentDispositionHeader.getFileName();
		// save the file to the server
		String output = "";
		try {
			saveFile(fileInputStream, filePath);
			output = "File saved to server location : " + filePath+" successfully";
		} catch (Exception e) {
			e.printStackTrace();
			output = "Unexpected server exception while uploading the file.";
		}
		return Response.status(200).entity(output).build();
	}

	@POST
	@Path("/upload3")
	@Consumes({MediaType.MULTIPART_FORM_DATA})
	public Response uploadFileWithData1(
			@FormDataParam("file") InputStream fileInputStream,
			@FormDataParam("file") FormDataContentDisposition cdh,
			@FormDataParam("emp") String emp) throws Exception {
		//http://localhost:8080/jerseyrest2x/itcresources/itc/upload3
		//Content-Type:multipart/form-data
		//form data as emp:abcd
		try {
			System.out.println("Emp data as String :::"+emp);
			//System.out.println("--->"+URLDecoder.decode(emp,"UTF-8"));
		} catch (Exception e) {
			e.printStackTrace();
		}

		String output = "";
		try {
			String filePath = Constants.SERVER_UPLOAD_LOCATION_FOLDER + cdh.getFileName();
			saveFile(fileInputStream, filePath);
			output = "File saved to server location : " + filePath+" successfully";
		} catch (Exception e) {
			e.printStackTrace();
			output = "Unexpected server exception while uploading the file.";
		}
		return Response.status(200).entity(output).build();
	}

	@Path("/download")
	@GET
	//	@Produces(MediaType.APPLICATION_OCTET_STREAM)
	@Produces("image/jpg")
	public Response getFile() {
		//GET http://localhost:8080/jerseyrest2x/itcresources/itc/download
		//You can invoke this url directly in the browser
		byte[] docStream = readContents("D:/temp/d.doc");
		return Response.ok(docStream, MediaType.APPLICATION_OCTET_STREAM)
				.header("content-disposition", "attachment; filename = r.jpg")
				.build();
	}

	private byte[] readContents(String filePath) {

		File file = new File(filePath);
		byte[] buffer = new byte[(int) file.length()];
		InputStream inStream = null;
		try {
			inStream = new FileInputStream(file);
```

```
                inStream.read(buffer);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
                    if (inStream != null)
                        inStream.close();
                } catch (Exception e2) {
                }
            }

            return buffer;
        }

        // save uploaded file to a defined location on the server
        private void saveFile(InputStream uploadedInputStream, String serverLocation) {

            try {
                OutputStream outpuStream = new FileOutputStream(new File(
                        serverLocation));
                int read = 0;
                byte[] bytes = new byte[1024];

                outpuStream = new FileOutputStream(new File(serverLocation));
                while ((read = uploadedInputStream.read(bytes)) != -1) {
                    outpuStream.write(bytes, 0, read);
                }
                outpuStream.flush();
                outpuStream.close();
            } catch (IOException e) {

                e.printStackTrace();
            }
        }
}
```

ITCAppConfig.java

```
package com.ddlab.rest.itc.resources;
import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
import org.glassfish.jersey.jackson.JacksonFeature;
import org.glassfish.jersey.media.multipart.MultiPartFeature;

@ApplicationPath("itcresources") //It becomes optional if you provide <url-pattern>/*</url-pattern> in web.xml
public class ITCAppConfig extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> resources = new HashSet<Class<?>>();
        resources.add(ITCInfotechServices.class);
        resources.add(MultiPartFeature.class);
        resources.add(JacksonFeature.class);
        return resources;
    }
}
```

Service Layer

EmployeeService.java

```
package com.ddlab.rest.service;
import com.ddlab.rest.entity.User;

public interface EmployeeService {

	public void createUser(User user);

	public void updateUser(User user);

	public void deleteUser(User user);

	public User getUserById(String id);

}
```

EmployeeServiceImpl.java

```
package com.ddlab.rest.service;
import com.ddlab.rest.entity.User;

public class EmployeeServiceImpl implements EmployeeService {

	public void createUser(User user) {
		System.out.println("User created successfully...");
	}

	public void updateUser(User user) {
		System.out.println("User information updated successfully...");
	}

	public void deleteUser(User user) {
		System.out.println("User information deleted successfully...");
	}

	public User getUserById(String id) {
		if( id == null ) throw new NullPointerException("User id can not be blank or empty");
		User user = new User();
		user.setPassword("");
		user.setUserName("Deb");
		user.setId(Integer.parseInt(id));
		return user;
	}
}
```

Entity Layer

Address.java

```
package com.ddlab.rest.entity;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.annotate.JsonProperty;

@XmlRootElement(name = "Adrs")
@XmlAccessorType(XmlAccessType.FIELD)
public class Address {
	@XmlElement(name = "DoorNo")
	@JsonProperty("DoorNo")
	private String doorNo;

	@XmlElement(name = "Street")
	@JsonProperty("Street")
	private String streetId;

	@XmlElement(name = "City")
	@JsonProperty("City")
	private String city;

	@XmlElement(name = "Country")
	@JsonProperty("Country")
	private String country;

	public String getDoorNo() {
		return doorNo;
	}
	public void setDoorNo(String doorNo) {
		this.doorNo = doorNo;
	}
	public String getStreetId() {
		return streetId;
	}

	public void setStreetId(String streetId) {
		this.streetId = streetId;
	}

	public String getCity() {
		return city;
	}

	public void setCity(String city) {
		this.city = city;
	}

	public String getCountry() {
		return country;
	}

	public void setCountry(String country) {
		this.country = country;
	}
}
```

Constants.java

```
package com.ddlab.rest.entity;
public interface Constants {
	public static final String SERVER_UPLOAD_LOCATION_FOLDER = "D:/temp/";
	public static final String DATA_LOCATION_FOLDER = "D:/temp/data";
}
```

Employee.java

```
package com.ddlab.rest.entity;
import java.io.IOException;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.annotate.JsonProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;

@XmlRootElement(name = "Emp")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee {

	@XmlElement(name = "Name")
	@JsonProperty("Name")
	private String name;

	@XmlElement(name = "Age")
	@JsonProperty("Age")
	private int age;

	@XmlElement(name = "Email")
	@JsonProperty("Email")
	private String emailId;

	@XmlElement(name = "Adrs")
	@JsonProperty("Adrs")
	private Address adrs;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getEmailId() {
		return emailId;
	}

	public void setEmailId(String emailId) {
		this.emailId = emailId;
```

```
	}

	public Address getAdrs() {
		return adrs;
	}

	public void setAdrs(Address adrs) {
		this.adrs = adrs;
	}

	public String toJSON() {
		ObjectMapper mapper = new ObjectMapper();
		try {
			return mapper.writeValueAsString(this);
		} catch (JsonGenerationException e) {
			e.printStackTrace();
		} catch (JsonMappingException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return null;
	}
}
```

FileUtil.java

```
package com.ddlab.rest.entity;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class FileUtil {

	public static void saveFile( String filePath , String contents ) {
		OutputStream out = null;
		try {
			out = new FileOutputStream(filePath);
			out.write(contents.getBytes());
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		finally {
			if( out != null ) {
				try {
					out.flush();
					out.close();
				} catch (IOException e) {
					e.printStackTrace();
				}
			}
		}

	}
}
```

User.java

```
package com.ddlab.rest.entity;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
import org.codehaus.jackson.annotate.JsonProperty;
import org.codehaus.jackson.annotate.JsonPropertyOrder;

@XmlRootElement(name = "User")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder={"userName", "password","id"})
@JsonPropertyOrder(value={"userName", "password", "id"})
public class User {

	@XmlElement(name = "username")
	@JsonProperty("username")
	private String userName;

	@XmlElement(name = "password")
	@JsonProperty("password")
	private String password;

	@XmlElement(name = "id")
	@JsonProperty("id")
	private int id;

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}
}
```

Maven(pom.xml)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>jersey-spring-rest</groupId>
	<artifactId>jersey-spring-rest</artifactId>
	<packaging>war</packaging>
	<version>0.0.1-SNAPSHOT</version>
	<name>jersey-spring-rest Maven Webapp</name>
	<url>http://maven.apache.org</url>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<org.springframework.version>3.2.1.RELEASE</org.springframework.version>
	</properties>

	<dependencies>
		<!-- Jersey related jar files -->
		<dependency>
			<groupId>javax.ws.rs</groupId>
			<artifactId>javax.ws.rs-api</artifactId>
			<version>2.0</version>
		</dependency>
		<dependency>
			<groupId>org.glassfish.jersey.containers</groupId>
			<artifactId>jersey-container-servlet</artifactId>
			<version>2.6</version>
		</dependency>
		<dependency>
			<groupId>org.glassfish.jersey.core</groupId>
			<artifactId>jersey-client</artifactId>
			<version>2.6</version>
		</dependency>
		<dependency>
			<groupId>org.glassfish.jersey.media</groupId>
			<artifactId>jersey-media-multipart</artifactId>
			<version>2.5.1</version>
		</dependency>
		<dependency>
			<groupId>org.glassfish.jersey.media</groupId>
			<artifactId>jersey-media-json-jackson</artifactId>
			<version>2.5.1</version>
		</dependency>
		<dependency>
			<groupId>org.codehaus.jackson</groupId>
			<artifactId>jackson-core-asl</artifactId>
			<version>1.9.13</version>
		</dependency>
		<dependency>
			<groupId>org.codehaus.jackson</groupId>
			<artifactId>jackson-mapper-asl</artifactId>
			<version>1.9.13</version>
		</dependency>
		<dependency>
			<groupId>javax.servlet</groupId>
```

```
			<artifactId>javax.servlet-api</artifactId>
			<version>3.1.0</version>
		</dependency>

		<!-- Jersey Spring Integration -->
		<dependency>
			<groupId>org.glassfish.jersey.ext</groupId>
			<artifactId>jersey-spring3</artifactId>
			<version>2.14</version>
		</dependency>
		<!-- Spring Jar files -->

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-beans</artifactId>
			<version>${org.springframework.version}</version>
		</dependency>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>${org.springframework.version}</version>
		</dependency>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>${org.springframework.version}</version>
		</dependency>

		<dependency>
			<groupId>commons-logging</groupId>
			<artifactId>commons-logging</artifactId>
			<version>1.2</version>
		</dependency>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>3.8.1</version>
			<scope>test</scope>
		</dependency>
	</dependencies>
	<build>
		<finalName>jersey-spring-rest</finalName>
	</build>
</project>
```

Spring Configuration (applicationContext.xml)

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
	http://www.springframework.org/schema/context
	http://www.springframework.org/schema/context/spring-context-3.0.xsd">

	<context:component-scan base-package="com.ddlab" />

	<bean id="dbHandler" class="com.ddlab.rest.service.LoginServiceImpl" />
</beans>
```

WEB.XML

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
	<display-name>Restful webservice with Jersey and Spring</display-name>
	<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>classpath:applicationContext.xml</param-value>
	</context-param>

	<listener>
		<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
	</listener>
	<servlet>
		<servlet-name>com.ddlab.rest.resources.ApplicationPkgs</servlet-name>
		<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-
class>
		<init-param>
			<param-name>javax.ws.rs.Application</param-name>
			<param-value>com.ddlab.rest.resources.ApplicationPkgs</param-value>
		</init-param>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>com.ddlab.rest.resources.ApplicationPkgs</servlet-name>
		<url-pattern>/*</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>
</web-app>
```

Index.jsp

```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

Java Code

ApplicationPkgs.java

```
package com.ddlab.rest.resources;
import org.glassfish.jersey.jackson.JacksonFeature;
import org.glassfish.jersey.server.ResourceConfig;

public class ApplicationPkgs extends ResourceConfig {
	public ApplicationPkgs() {
		super(LoginServiceResource.class,JacksonFeature.class);
	}
}
```

BaseResource.java

```
package com.ddlab.rest.resources;
import java.nio.file.AccessDeniedException;
import javax.ws.rs.NotFoundException;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

public class BaseResource {

	protected WebApplicationException createWebappException(Exception incomingException) {
		Status status;
		String string = null;
		if (incomingException instanceof SecurityException || incomingException instanceof
AccessDeniedException) {
			status = Status.FORBIDDEN;
		} else if (incomingException instanceof IllegalArgumentException) {
			status = Status.BAD_REQUEST;
			string=incomingException.getMessage();
		} else if (incomingException instanceof NotFoundException) {
			status = Status.NOT_FOUND;
		} else if (incomingException instanceof Exception) {
			status = Status.INTERNAL_SERVER_ERROR;
		} else {
			status = Status.INTERNAL_SERVER_ERROR;
		}
		return new
WebApplicationException(Response.status(status).entity(string).type("text/plain").build());
	}
}
```

LoginServiceResource.java

```
package com.ddlab.rest.resources;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.Response.Status;
import org.springframework.beans.factory.annotation.Autowired;
import com.ddlab.rest.entity.User;
import com.ddlab.rest.service.LoginService;
```

```
@Path("1/loginservice")
public class LoginServiceResource extends BaseResource {

	@Context
	HttpServletRequest request;

	@Context
	ServletConfig servletConfig;

	@Autowired
	private LoginService loginService;

	@Path("/create")
	@POST
	@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public Response createUser(User user) {
		ResponseBuilder responseBuilder = null;
		try {
			if(user.getUserName() == null || user.getPassword() == null )
				responseBuilder = Response.status(Status.BAD_REQUEST).entity("Incorrect
password");
			loginService.createUser(user);
			responseBuilder = Response.status(Status.CREATED).entity("User created
successfully");
		} catch (Exception e) {
			responseBuilder = Response.serverError();
		}
		return responseBuilder.build();
	}

	@Path("/update")
	@PUT
	@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public void updateUser(User user) { //204 No content
		if(user.getUserName() == null || user.getPassword() == null )
			throw createWebappException(new IllegalArgumentException("Incorrect credentials"));
		loginService.updateUser(user);
	}

	@Path("/delete")
	@DELETE
	@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public String deleteUser(User user) { //204 No content
		if(user.getUserName() == null || user.getPassword() == null )
			throw createWebappException(new IllegalArgumentException("Incorrect credentials"));
		loginService.deleteUser(user);
		return "User "+user.getUserName()+" has been removed from the system successfully";
	}

	@Path("/userid")
	@GET
	@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public User getUser(@QueryParam("id") int id) { //204 No content
		if(id == 0 )
			throw createWebappException(new IllegalArgumentException("Incorrect ID"));
		return loginService.getUserById(String.valueOf(id));
	}
}
```

LoginService.java

```
package com.ddlab.rest.service;
import com.ddlab.rest.entity.User;

public interface LoginService {

	public void createUser(User user);

	public void updateUser(User user);

	public void deleteUser(User user);

	public User getUserById(String id);
}
```

LoginServiceImpl.java

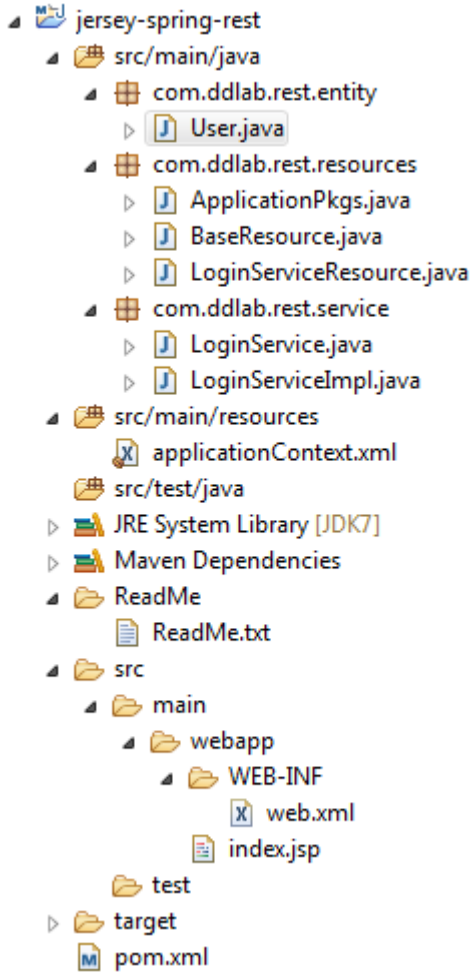
```
package com.ddlab.rest.service;
import com.ddlab.rest.entity.User;

public class LoginServiceImpl implements LoginService {

	public void createUser(User user) {
		System.out.println("User created successfully...");
	}

	public void updateUser(User user) {
		System.out.println("User information updated successfully...");
	}

	public void deleteUser(User user) {
		System.out.println("User information deleted successfully...");
	}

	public User getUserById(String id) {
		if( id == null ) throw new NullPointerException("User id can not be blank or empty");
		User user = new User();
		user.setPassword("");
		user.setUserName("Deb");
		user.setId(Integer.parseInt(id));
		return user;
	}
}
```

User.java

```
package com.ddlab.rest.entity;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
import org.codehaus.jackson.annotate.JsonProperty;
import org.codehaus.jackson.annotate.JsonPropertyOrder;
```

```
@XmlRootElement(name = "User")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder={"userName", "password","id"})
@JsonPropertyOrder(value={"userName", "password", "id"})
public class User {
	@XmlElement(name = "username")
	@JsonProperty("username")
	private String userName;

	@XmlElement(name = "password")
	@JsonProperty("password")
	private String password;

	@XmlElement(name = "id")
	@JsonProperty("id")
	private int id;

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}
}
```

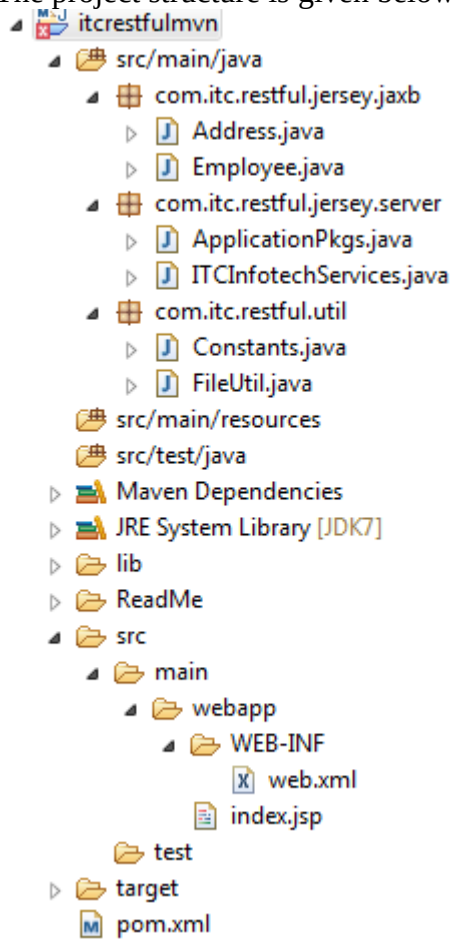
Project Structure



RESTful Web service using Jersey 2.6

Project Name : itcrestfulmvn

The project structure is given below.



Constants.java

```
package com.itc.restful.util;
public interface Constants {

	public static final String SERVER_UPLOAD_LOCATION_FOLDER = "D:/temp/";
	public static final String DATA_LOCATION_FOLDER = "D:/temp/data";

}
```

FileUtil.java

```
package com.itc.restful.util;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class FileUtil {

	public static void saveFile( String filePath , String contents ) {
		OutputStream out = null;
		try {
			out = new FileOutputStream(filePath);
			out.write(contents.getBytes());
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		finally {
			if( out != null ) {
```

```
                        try {
                              out.flush();
                              out.close();
                        } catch (IOException e) {
                              e.printStackTrace();
                        }
                  }
            }

      }

}
```

Employee.java

```
package com.itc.restful.jersey.jaxb;
import java.io.IOException;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.annotate.JsonProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;

@XmlRootElement(name = "Emp")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee {

      @XmlElement(name = "Name")
      @JsonProperty("Name")
      private String name;

      @XmlElement(name = "Age")
      @JsonProperty("Age")
      private int age;

      @XmlElement(name = "Email")
      @JsonProperty("Email")
      private String emailId;

      @XmlElement(name = "Adrs")
      @JsonProperty("Adrs")
      private Address adrs;

      public String getName() {
            return name;
      }

      public void setName(String name) {
            this.name = name;
      }

      public int getAge() {
            return age;
      }

      public void setAge(int age) {
            this.age = age;
      }

      public String getEmailId() {
            return emailId;
      }
```

```
	public void setEmailId(String emailId) {
		this.emailId = emailId;
	}

	public Address getAdrs() {
		return adrs;
	}

	public void setAdrs(Address adrs) {
		this.adrs = adrs;
	}

	public String toJSON() {
		ObjectMapper mapper = new ObjectMapper();
		try {
			return mapper.writeValueAsString(this);
		} catch (JsonGenerationException e) {
			e.printStackTrace();
		} catch (JsonMappingException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return null;
	}

}
```

Address.java

```
package com.itc.restful.jersey.jaxb;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.annotate.JsonProperty;

@XmlRootElement(name = "Adrs")
@XmlAccessorType(XmlAccessType.FIELD)
public class Address {

	@XmlElement(name = "DoorNo")
	@JsonProperty("DoorNo")
	private String doorNo;

	@XmlElement(name = "Street")
	@JsonProperty("Street")
	private String streetId;

	@XmlElement(name = "City")
	@JsonProperty("City")
	private String city;

	@XmlElement(name = "Country")
	@JsonProperty("Country")
	private String country;

	public String getDoorNo() {
		return doorNo;
	}

	public void setDoorNo(String doorNo) {
		this.doorNo = doorNo;
	}

	public String getStreetId() {
```

```
		return streetId;
	}

	public void setStreetId(String streetId) {
		this.streetId = streetId;
	}

	public String getCity() {
		return city;
	}

	public void setCity(String city) {
		this.city = city;
	}

	public String getCountry() {
		return country;
	}

	public void setCountry(String country) {
		this.country = country;
	}

}
```

ApplicationPkgs.java

```
package com.itc.restful.jersey.server;
import org.glassfish.jersey.jackson.JacksonFeature;
import org.glassfish.jersey.server.ResourceConfig;

public class ApplicationPkgs extends ResourceConfig {

	public ApplicationPkgs() {
		super(ITCInfotechServices.class,JacksonFeature.class);
	}

}
```

ITCInfotechServices.java

```
package com.itc.restful.jersey.server;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
```

```
import javax.ws.rs.core.Response;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.glassfish.jersey.media.multipart.ContentDisposition;
import org.glassfish.jersey.media.multipart.FormDataBodyPart;
import org.glassfish.jersey.media.multipart.FormDataMultiPart;
import com.itc.restful.jersey.jaxb.Employee;
import com.itc.restful.util.Constants;
import com.itc.restful.util.FileUtil;

@Path("itc")
public class ITCInfotechServices {

	@Context
	HttpServletRequest request;

	@Context
	ServletConfig servletConfig;

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getOrganisationName() {
		return "ITC Infotech, Bangalore, Karnataka";
	}

	@Path("/address")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddress() {
		// URL : http://localhost:8080/itcrestful/itc/address
		return "ITC Infotech India Limited, 18, Banaswadi Main Rd, Maruthi Sevanagar,
Bangalore, 560005";
	}

	// ~~~~~~~~~~~~~~~~~~~~~~ @PathParam ~~~~~~~~~~~~~~~~~~~~~~~~~~
	@Path("/address/{areaCode}")
	// USA,EUROPE,ASIA
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddressByCode(@PathParam("areaCode") String areaCode) {
		// http://localhost:8080/itcrestful/itc/address/USA
		// http://localhost:8080/itcrestful/itc/address/europe
		try {
			if (areaCode.equalsIgnoreCase("USA"))
				return "12 North State, Route 17,Suite 303,Paramus,New Jersey,NJ-
07652";
			else if (areaCode.equalsIgnoreCase("Europe"))
				return "Newell Consulting Oy,P.O. Box 16 , Olari,02211 Espoo,
Helsinki";
			else if (areaCode.equalsIgnoreCase("Africa"))
				return "Johannesburg,2nd Floor, West Tower,Nelson Mandela Square,Maude
Street, Sandton,Johannesburg, 2196";
			else if (areaCode.equalsIgnoreCase("Asia"))
				return "ITC Infotech India Limited, 18, Banaswadi Main Rd, Maruthi
Sevanagar, Bangalore, 560005";
			else
				return "No such area code exists for ITC";
		} catch (Exception e) {
			return "No such area code exists for ITC";
		}
	}

	// ~~~~~~~~~~~~~~~~~~~~~~ @QueryParam ~~~~~~~~~~~~~~~~~~~~~~~~~~
	@Path("/regionaladdress/{areaCode}")
```

```
	// USA,EUROPE,ASIA
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddressByCountry(@PathParam("areaCode") String areaCode,
			@QueryParam("country") String country) {
		// http://localhost:8080/itcrestful/itc/regionaladdress/Europe?country=FI
		try {
			if (areaCode.equalsIgnoreCase("USA")
					&& country.equalsIgnoreCase("NJ"))
				return "12 North State, Route 17,Suite 303,Paramus,New Jersey,NJ-
07652";
			else if (areaCode.equalsIgnoreCase("Europe")
					&& country.equalsIgnoreCase("FI"))
				return "Newell Consulting Oy,P.O. Box 16 , Olari,02211 Espoo,
Helsinki";
			else if (areaCode.equalsIgnoreCase("Europe")
					&& country.equalsIgnoreCase("SE"))
				return "C/o Matrisen AB,Box 22059 , 104 22 Stockholm";
			else if (areaCode.equalsIgnoreCase("Europe")
					&& country.equalsIgnoreCase("DK"))
				return "Havnegade 39, 3. sal,1058 Copenhagen K";
			else if (areaCode.equalsIgnoreCase("Asia")
					&& country.equalsIgnoreCase("IN"))
				return "ITC Infotech India Limited, 18, Banaswadi Main Rd, Maruthi
Sevanagar, Bangalore, 560005";
			else
				return "No such area code exists for ITC";
		} catch (Exception e) {
			return "No such area code exists for ITC";
		}
	}

	// ~~~~~~~~~~~~~~~~~~~~~~~~ @MatrixParam ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	// /books/2011;author=mkyong;country=malaysia”

	@Path("/itcaddress")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getITCAddress(@MatrixParam("country") String country,
			@MatrixParam("areacode") String areaCode) {
		// http://localhost:8080/itcrestful/itc/itcaddress;country=FI;areacode=europe
		return getAddressByCountry(areaCode, country);
	}

	// ~~~~~~~~~~~~~~~~~~~~~~~~ @FormParam ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	@Path("/postaddress")
	@POST
	@Produces(MediaType.TEXT_PLAIN)
	public String postNGetITCAddress(@FormParam("country") String country,
			@FormParam("areacode") String areaCode) {
		// http://localhost:8080/itcrestful/itc/postaddress
		return getAddressByCountry(areaCode, country);
	}

	// ~~~~~~~~~~~~~~~~~~~~~~~~ @PathParam with Object ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	@Path("/emp/{id}")
	@GET
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public Response getEmployee(@PathParam("id") String id) {
//		Address adrs = new Address();
//		adrs.setCity("Bangalore");
//		adrs.setCountry("Karnataka");
//		adrs.setDoorNo("12-A");
//		adrs.setStreetId("Street-11");
//
```

```
//          Employee emp = new Employee();
//          emp.setAdrs(adrs);
//          emp.setAge(23);
//          emp.setEmailId("deba@gmail.com");
//          emp.setName("Deba");
            ObjectMapper mapper = new ObjectMapper();
            Employee emp = null;
            try {
                  File file = new
File(Constants.DATA_LOCATION_FOLDER+File.separator+id+".json");
                  if( ! file.exists() )
                        return Response.status(404).entity("No information found for the given
employee id ...").build();
                  emp = mapper.readValue(file, Employee.class);
            } catch (JsonParseException e) {
                  e.printStackTrace();
            } catch (JsonMappingException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            // return emp;
            return Response.ok(emp).build();
      }

      @Path("/createemp")
      @POST
      @Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      public Response createEmp(Employee emp) {
            System.out.println("Employee object received successfully .........");
            //store into file system
            if( new
File(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json").exists()) {
                  return Response.status(409).entity("Employee already exists in the system
...").build();
            }

      FileUtil.saveFile(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json",
emp.toJSON());
            return Response.ok("Employee created successfully...").build();
      }

      @Path("/updateemp")
      @PUT
      @Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
      public Response updateEmp(Employee emp) {
            System.out.println("Employee object received successfully .........");
            //store into file system
            if( new
File(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json").exists()) {

      FileUtil.saveFile(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json",
emp.toJSON());
                  return Response.ok("Employee info updated successfully...").build();

            }
            else
                  return Response.status(404).entity("Employee information not found, hence can
not be updated ...").build();
      }

      @Path("/deleteemp")
      @DELETE
```

```
	@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public Response deleteEmp(Employee emp) {
		System.out.println("Employee object received successfully .........");
		//store into file system
		File file = new
File(Constants.DATA_LOCATION_FOLDER+File.separator+emp.getName()+".json");
		if( file.exists()) {
			file.delete();
			return Response.ok("Employee info deleted successfully...").build();

		}
		else
			return Response.status(404).entity("Employee information not found, hence can
not be deleted ...").build();
	}

	@POST
	@Path("/upload")
	@Consumes(MediaType.MULTIPART_FORM_DATA)
	public Response uploadFile(FormDataMultiPart form) {
		FormDataBodyPart filePart = form.getField("file");
		ContentDisposition headerOfFilePart = filePart.getContentDisposition();
		InputStream fileInputStream = filePart.getValueAs(InputStream.class);
		String filePath = Constants.SERVER_UPLOAD_LOCATION_FOLDER
				+ headerOfFilePart.getFileName();
		// save the file to the server
		saveFile(fileInputStream, filePath);
		String output = "File saved to server location using FormDataMultiPart : "
				+ filePath;
		try {
			fileInputStream.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return Response.status(200).entity(output).build();
	}

	@Path("/download")
	@GET
//	@Produces(MediaType.APPLICATION_OCTET_STREAM)
	@Produces("image/jpg")
	public Response getFile() {

		byte[] docStream = readContents("D:/temp/d.doc");

		return Response.ok(docStream, MediaType.APPLICATION_OCTET_STREAM)
				.header("content-disposition", "attachment; filename = r.jpg")
				.build();
	}

	private byte[] readContents(String filePath) {

		File file = new File(filePath);
		byte[] buffer = new byte[(int) file.length()];
		InputStream inStream = null;
		try {
			inStream = new FileInputStream(file);
			inStream.read(buffer);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			try {
```

```
                        if (inStream != null)
                              inStream.close();
                  } catch (Exception e2) {
                  }
            }

            return buffer;

      }

      // save uploaded file to a defined location on the server
      private void saveFile(InputStream uploadedInputStream, String serverLocation) {

            try {
                  OutputStream outpuStream = new FileOutputStream(new File(
                              serverLocation));
                  int read = 0;
                  byte[] bytes = new byte[1024];

                  outpuStream = new FileOutputStream(new File(serverLocation));
                  while ((read = uploadedInputStream.read(bytes)) != -1) {
                        outpuStream.write(bytes, 0, read);
                  }
                  outpuStream.flush();
                  outpuStream.close();
            } catch (IOException e) {

                  e.printStackTrace();
            }
      }

}
```

web.xml

```
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd">
    <servlet>
        <servlet-name>com.itc.restful.jersey.server.ApplicationPkgs</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>javax.ws.rs.Application</param-name>
            <param-value>com.itc.restful.jersey.server.ApplicationPkgs</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>com.itc.restful.jersey.server.ApplicationPkgs</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Dependencies in pom.xml

```
<dependencies>
            <dependency>
                  <groupId>junit</groupId>
                  <artifactId>junit</artifactId>
                  <version>3.8.1</version>
                  <scope>test</scope>
            </dependency>
            <dependency>
                  <groupId>javax.ws.rs</groupId>
                  <artifactId>javax.ws.rs-api</artifactId>
```

```
                <version>2.0</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.containers</groupId>
                <artifactId>jersey-container-servlet</artifactId>
                <version>2.6</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.core</groupId>
                <artifactId>jersey-client</artifactId>
                <version>2.6</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>org.codehaus.jackson</groupId>
                <artifactId>jackson-core-asl</artifactId>
                <version>1.9.13</version>
            </dependency>
            <dependency>
                <groupId>org.codehaus.jackson</groupId>
                <artifactId>jackson-mapper-asl</artifactId>
                <version>1.9.13</version>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.media</groupId>
                <artifactId>jersey-media-multipart</artifactId>
                <version>2.5.1</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>3.1.0</version>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.media</groupId>
                <artifactId>jersey-media-json-jackson</artifactId>
                <version>2.5.1</version>
                <scope>provided</scope>
            </dependency>
</dependencies>
```

ReadMe.txt

WADL URL : http://localhost:8080/itcrestful/application.wadl
General URL : http://localhost:8080/itcrestful/itc
Address URL : http://localhost:8080/itcrestful/itc/address
US Address : http://localhost:8080/itcrestful/itc/address/USA
EU Address : http://localhost:8080/itcrestful/itc/address/europe

By Country Code

http://localhost:8080/itcrestful/itc/regionaladdress/Europe?country=FI

Using Matrix Param

http://localhost:8080/itcrestful/itc/itcaddress;country=FI;areacode=europe

http://localhost:8080/itcrestful/itc/postaddress

Click on x-www-form-urlencoded in Postman
country - FI
areacode - europe

For Emp Id
http://localhost:8080/itcrestful/itc/emp/id123
in the header, pass the following

Accept - application/xml or application/json
Select GET request

Emp creation
http://localhost:8080/itcrestful/itc/createemp
For Emp creation

```
<Emp>
    <Name>Deba</Name>
    <Age>23</Age>
    <Email>deba@gmail.com</Email>
    <Adrs>
        <DoorNo>12-A</DoorNo>
        <Street>Street-11</Street>
        <City>Bangalore</City>
        <Country>Karnataka</Country>
    </Adrs>
</Emp>
```

or

```
{
    "Name": "Deba",
    "Age": 23,
    "Email": "deba@gmail.com",
    "Adrs": {
        "DoorNo": "12-A",
        "Street": "Street-11",
        "City": "Bangalore",
        "Country": "Karnataka"
    }
}
```

Accept - application/xml or application/json
Content-Type ---------- application/xml or application/json
click on Raw to provide the data and down click on Pretty

For Uploading binary file
---
http://localhost:8080/itcrestful/itc/upload

File Upload and Download using Jersey RESTful web service

Project Name : downloadupload

ApplicationPkgs.java

```
package com.itc.restful.jersey.files;
import org.glassfish.jersey.media.multipart.MultiPartFeature;
import org.glassfish.jersey.server.ResourceConfig;

public class ApplicationPkgs extends ResourceConfig {

	public ApplicationPkgs() {
		super(FileUploadNDownload.class,MultiPartFeature.class);
	}

}
```

FileUploadNDownload.java

```
package com.itc.restful.jersey.files;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.glassfish.jersey.media.multipart.FormDataContentDisposition;
import org.glassfish.jersey.media.multipart.FormDataParam;

@Path("/files")
public class FileUploadNDownload {

	private static final String SERVER_UPLOAD_LOCATION_FOLDER =
"D:"+File.separator+"temp"+File.separator;
	private static final String SERVER_DOWNLOAD_LOCATION_FOLDER =
"D:"+File.separator+"temp"+File.separator;
	private static final String FILE_NAME = "d.doc";
	private static final String IMG_FILE_NAME = "r.jpg";

	@POST
	@Path("/upload")
	@Consumes(MediaType.MULTIPART_FORM_DATA)
	public Response uploadFile( @FormDataParam("file") InputStream fileInputStream,
@FormDataParam("file") FormDataContentDisposition contentDispositionHeader) {
//	public Response uploadFile( @FormDataParam("file") String s, @FormDataParam("file")
FormDataContentDisposition contentDispositionHeader) {
		String filePath = SERVER_UPLOAD_LOCATION_FOLDER +
contentDispositionHeader.getFileName();
		// save the file to the server
		String output = "";
		try {
//			InputStream fileInputStream = new FileInputStream(s);
			FileUtil.saveFile(fileInputStream, filePath);
			output = "File saved to server location : " + filePath+" successfully";
		} catch (Exception e) {
			e.printStackTrace();
			output = "Unexpected server exception while uploading the file.";
		}
		return Response.status(200).entity(output).build();
	}

	@Path("/download")
```

```
	@GET
	@Produces(MediaType.APPLICATION_OCTET_STREAM)
	public Response getFile() {
		byte[] docStream =
FileUtil.readContents(SERVER_DOWNLOAD_LOCATION_FOLDER+FILE_NAME);
		String attachementStr =  "attachment; filename = "+FILE_NAME;
		return Response
				.ok(docStream, MediaType.APPLICATION_OCTET_STREAM)
				.header("content-disposition",attachementStr)
				.build();
	}

	@Path("/download/img")
	@GET
	@Produces(MediaType.APPLICATION_OCTET_STREAM)
	public Response downloadImage() {
		//http://localhost:8080/restfulbin/rest/files/download/img
		String attachementStr =  "attachment; filename = "+IMG_FILE_NAME;
		File file = new File(SERVER_DOWNLOAD_LOCATION_FOLDER+IMG_FILE_NAME);
		return Response.ok((Object) file).header("content-
disposition",attachementStr).build();
	}

	@Path("/show/image")
	@GET
	@Produces("image/png")
	public Response showImage() {

		InputStream inStream = null;
		try {
			inStream = new
FileInputStream(SERVER_DOWNLOAD_LOCATION_FOLDER+IMG_FILE_NAME);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
		return Response.ok().entity(inStream).build();
	}
}
```

FileUtil.java

```
package com.itc.restful.jersey.files;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class FileUtil {

	public static byte[] readContents(String filePath) {
		File file = new File(filePath);
		byte[] buffer = new byte[(int) file.length()];
		InputStream inStream = null;
		try {
			inStream = new FileInputStream(file);
			inStream.read(buffer);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		finally {
```

```
                try {
                      if( inStream != null ) inStream.close();
                } catch (Exception e2) {
                }
          }
          return buffer;
    }

    public static byte[] readContents1(String filePath) {

          InputStream inStream = null;
          ByteArrayOutputStream outStream = null;
          try {
                inStream = new FileInputStream(filePath);
                outStream = new ByteArrayOutputStream();
                int c;
                while( (c = inStream.read()) != -1 ) {
                      outStream.write(c);
                }
          } catch (FileNotFoundException e) {
                e.printStackTrace();
          } catch (IOException e) {
                e.printStackTrace();
          }
          finally {
                try {
                      if( inStream != null ) inStream.close();
                      if( outStream != null ) outStream.close();
                } catch (Exception e2) {
                }
          }
          return outStream.toByteArray();
    }

    public static void saveFile(InputStream uploadedInputStream, String serverLocation)
throws Exception {
          OutputStream outStream = null;
          outStream = new FileOutputStream(serverLocation);
          int read = 0;
          byte[] buffer = new byte[1024];
          while( (read = uploadedInputStream.read(buffer)) != -1 ) {
                outStream.write(buffer,0,read);
          }
          if( outStream != null) {
                outStream.flush();
                outStream.close();
          }
    }

}
```

web.xml

```
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <!-- <servlet>
          <servlet-name>FileUploadNDownload</servlet-name>
          <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-
class>
          <init-param>
                <param-name>com.sun.jersey.config.property.packages</param-name>
```

```
                <param-value>com.itc.restful.jersey.files</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
      </servlet> -->
      <servlet>
        <servlet-name>FileUploadNDownload</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>javax.ws.rs.Application</param-name>
            <param-value>com.itc.restful.jersey.files.ApplicationPkgs</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
      <servlet-mapping>
            <servlet-name>FileUploadNDownload</servlet-name>
            <url-pattern>/rest/*</url-pattern>
      </servlet-mapping>
      <welcome-file-list>
            <welcome-file>form.html</welcome-file>
      </welcome-file-list>
</web-app>
```

form.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Form Page</title>
</head>
<body>
<h1>Upload a File</h1>

      <form action="rest/files/upload" method="post" enctype="multipart/form-data">

         <p>
            Select a file : <input type="file" name="file" size="45" />
         </p>

         <input type="submit" value="Upload It" />
      </form>

</body>
</html>
```

ReadMe.txt

```
WADL URL : http://localhost:8080/restfulbin/rest/application.wadl
REST URL for Upload : http://localhost:8080/restfulbin/rest/files/upload



REST URL for Download doc : http://localhost:8080/restfulbin/rest/files/download
REST URL for Download image : http://localhost:8080/restfulbin/rest/files/download/img
REST URL to Show image : http://localhost:8080/restfulbin/rest/files/show/image

Page access url : http://localhost:8080/restfulbin/
```

Required Portion in pom.xml

```
      <repositories>
            <repository>
                  <id>snapshot-repository.java.net</id>
                  <name>Java.net Snapshot Repository for Maven</name>
                  <url>https://maven.java.net/content/repositories/snapshots/</url>
                  <layout>default</layout>
            </repository>
      </repositories>
```

```
<dependencies>
	<dependency>
		<groupId>junit</groupId>
		<artifactId>junit</artifactId>
		<version>3.8.1</version>
		<scope>test</scope>
	</dependency>
	<dependency>
		<groupId>javax.ws.rs</groupId>
		<artifactId>javax.ws.rs-api</artifactId>
		<version>2.0</version>
		<scope>provided</scope>
	</dependency>

	<dependency>
		<groupId>com.sun.jersey</groupId>
		<artifactId>jersey-server</artifactId>
		<version>1.17.1</version>
	</dependency>
	<dependency>
		<groupId>com.sun.jersey</groupId>
		<artifactId>jersey-core</artifactId>
		<version>1.17.1</version>
	</dependency>
	<dependency>
		<groupId>com.sun.jersey</groupId>
		<artifactId>jersey-servlet</artifactId>
		<version>1.17.1</version>
	</dependency>
	<dependency>
		<groupId>org.glassfish.jersey.media</groupId>
		<artifactId>jersey-media-multipart</artifactId>
		<version>2.5.1</version>
		<scope>provided</scope>
	</dependency>
	<dependency>
		<groupId>javax.servlet</groupId>
		<artifactId>javax.servlet-api</artifactId>
		<version>3.1.0</version>
	</dependency>
</dependencies>
```

Introduction

You can use Spring framework to define your REST implementation. The following annotations are used.
@Controller – Used in the class level
@RequestMapping("/emp") – Used in the class level as the root path
@RequestMapping(value = "ID/{id}", method = RequestMethod.*GET*) - Used in method level as the sub path
@ResponseBody – Used in the method leve
@PathVariable String id – Used in the method parameter
@RequestParam("id") String id – used in the method parameter to pass id as request parameter.

```
@RequestMapping(value = "create", method = RequestMethod.POST,
                consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE },
                produces = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
```

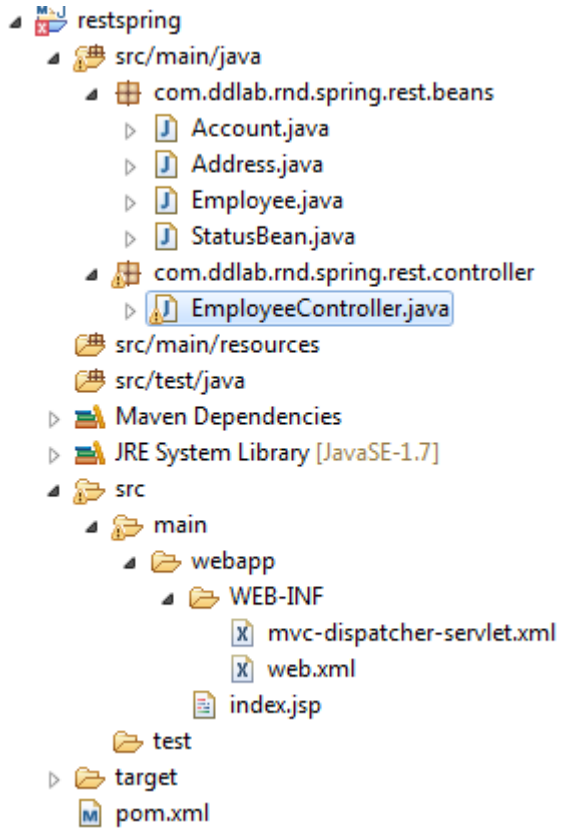
The above annotation defines that the server can accept both JSON and XML request as request body, to specify you to provide the value as follows.
consumes = {MediaType.*APPLICATION_JSON_VALUE*, MediaType.*APPLICATION_XML_VALUE* }

Similarly it provides information to produce the result in the XML or JSON format, to specify you have to give the values as
produces = {MediaType.*APPLICATION_JSON_VALUE*, MediaType.*APPLICATION_XML_VALUE* }

@ResponseStatus(HttpStatus.*CREATED*) – Used in the method level as Created
@ResponseBody - spring will try to convert its return value and write it to the http response automatically
@RequestBody Employee emp - Used in method parameter level. Spring will try to convert the content of the incoming request body to your parameter object on the fly.

Remember that Spring controller can accept both XML or JSON request body or either or.

The complete project is given below. The name of the project "restspring" . The following jar files are used.
aopalliance-1.0.jar
commons-logging-1.1.1.jar
jackson-core-asl-1.9.10.jar
jackson-mapper-asl-1.9.10.jar
servlet-api-2.5.jar
spring-aop-3.2.2.RELEASE.jar
spring-beans-3.2.2.RELEASE.jar
spring-context-3.2.2.RELEASE.jar
spring-core-3.2.2.RELEASE.jar
spring-expression-3.2.2.RELEASE.jar
spring-web-3.2.2.RELEASE.jar
spring-webmvc-3.2.2.RELEASE.jar



The following beans are given below.

Account.java

```
package com.ddlab.rnd.spring.rest.beans;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.annotate.JsonProperty;

@XmlRootElement
public class Account {

	@JsonProperty(value="AccountName") private String name;
	@JsonProperty(value="AccountNo") private String actNo;

	@XmlElement
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	@XmlElement
	public String getActNo() {
		return actNo;
	}
	public void setActNo(String actNo) {
		this.actNo = actNo;
	}
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((actNo == null) ? 0 : actNo.hashCode());
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}
```

```
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Account other = (Account) obj;
		if (actNo == null) {
			if (other.actNo != null)
				return false;
		} else if (!actNo.equals(other.actNo))
			return false;
		if (name == null) {
			if (other.name != null)
				return false;
		} else if (!name.equals(other.name))
			return false;
		return true;
	}
	@Override
	public String toString() {
		return "Account [name=" + name + ", actNo=" + actNo + "]";
	}
}
```

Address.java

```
package com.ddlab.rnd.spring.rest.beans;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.annotate.JsonProperty;

@XmlRootElement
public class Address {

	@JsonProperty(value="StreetName") private String streetName;
	@JsonProperty(value="CountryName") private String countryName;

	@XmlElement
	public String getStreetName() {
		return streetName;
	}
	public void setStreetName(String streetName) {
		this.streetName = streetName;
	}
	@XmlElement
	public String getCountryName() {
		return countryName;
	}
	public void setCountryName(String countryName) {
		this.countryName = countryName;
	}
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result
				+ ((countryName == null) ? 0 : countryName.hashCode());
		result = prime * result
				+ ((streetName == null) ? 0 : streetName.hashCode());
		return result;
	}
	@Override
	public boolean equals(Object obj) {
```

```
            if (this == obj)
                  return true;
            if (obj == null)
                  return false;
            if (getClass() != obj.getClass())
                  return false;
            Address other = (Address) obj;
            if (countryName == null) {
                  if (other.countryName != null)
                        return false;
            } else if (!countryName.equals(other.countryName))
                  return false;
            if (streetName == null) {
                  if (other.streetName != null)
                        return false;
            } else if (!streetName.equals(other.streetName))
                  return false;
            return true;
      }
      @Override
      public String toString() {
            return "Address [streetName=" + streetName + ", countryName="
                        + countryName + "]";
      }
}
```

Employee.java

```
package com.ddlab.rnd.spring.rest.beans;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.annotate.JsonProperty;
import org.codehaus.jackson.map.annotate.JsonRootName;

@XmlRootElement
@JsonRootName(value="Emp")
public class Employee {

      @JsonProperty(value="Id") private String id;
      @JsonProperty(value="Name") private String name;
      @JsonProperty(value="Adrs") private Address adrs;
      @JsonProperty(value="Account") private Account act;

      public Employee( String id )
      {
            this.id = id;
      }

      public Employee()
      {

      }

      @XmlElement
      public Address getAdrs() {
            return adrs;
      }
      public void setAdrs(Address adrs) {
            this.adrs = adrs;
      }
      @XmlElement
      public Account getAct() {
            return act;
      }
      public void setAct(Account act) {
            this.act = act;
      }
```

```
	@XmlElement
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	@XmlElement
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 : id.hashCode());
		return result;
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Employee other = (Employee) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}
	@Override
	public String toString() {
		return "Employee [id=" + id + ", name=" + name + ", adrs=" + adrs
				+ ", act=" + act + "]";
	}
}
```

Status.java

```
package com.ddlab.rnd.spring.rest.beans;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="Status")
public class StatusBean {

	private String status = "Failed";
	private String errCode = "";
	private String errMsg = "";

	@XmlElement(defaultValue="Status")
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}
```

```
	@XmlElement(defaultValue="ErrorCode")
	public String getErrCode() {
		return errCode;
	}
	public void setErrCode(String errCode) {
		this.errCode = errCode;
	}

	@XmlElement(defaultValue="ErrorMessage")
	public String getErrMsg() {
		return errMsg;
	}
	public void setErrMsg(String errMsg) {
		this.errMsg = errMsg;
	}
}
```

EmployeeController.java

```
package com.ddlab.rnd.spring.rest.controller;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import com.ddlab.rnd.spring.rest.beans.Employee;
import com.ddlab.rnd.spring.rest.beans.StatusBean;

@Controller
@RequestMapping("/emp")
public class EmployeeController {

	private List<Employee> empList = new ArrayList<Employee>();

	@RequestMapping(value = "ID/{id}", method = RequestMethod.GET)
	@ResponseBody
	public Object getEmp( @PathVariable String id )
	{
		Object obj = null;
		Employee emp = new Employee(id);

		if( empList.contains(emp))
		{
			for( Employee emp1 : empList )
			{
				if( emp.equals(emp1))
				{
					obj = emp1;
					break;
				}
			}
		}
```

```
		else
		{
			StatusBean statusBean = new StatusBean();
			statusBean.setErrCode("002");
			statusBean.setErrMsg("Employee with this id "+id+" does not exist");
			obj = statusBean;
		}

		return obj;
	}

	@RequestMapping(value = "ID", method = RequestMethod.GET)
	@ResponseBody
	public Object getEmp1( @RequestParam("id") String id )
	{
		/*Request will be like this
		 * http://localhost:8080/restspring/emp/ID?id=Id123
		 */
		System.out.println("------- Invoking method getEmp1 with @RequestParam as parameter
------");
		Object obj = null;
		Employee emp = new Employee(id);

		if( empList.contains(emp))
		{
			for( Employee emp1 : empList )
			{
				if( emp.equals(emp1))
				{
					obj = emp1;
					break;
				}
			}
		}
		else
		{
			StatusBean statusBean = new StatusBean();
			statusBean.setErrCode("002");
			statusBean.setErrMsg("Employee with this id "+id+" does not exist");
			obj = statusBean;
		}
		return obj;
	}

	@RequestMapping(value = "create", method = RequestMethod.POST,
	consumes ={MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }, produces
	= {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
	@ResponseStatus(HttpStatus.CREATED)
	@ResponseBody
	public StatusBean create( @RequestBody Employee emp ) {
		StatusBean statusBean = new StatusBean();
		if( empList.contains(emp) )
		{
			statusBean.setErrCode("001");
			statusBean.setErrMsg("Employee with this id "+emp.getId()+" already exists");
		}
		else
		{
			empList.add(emp);
			statusBean.setStatus("Success");
		}

		display();
		return statusBean;
	}
```

```
	@RequestMapping(value = "create1", method = RequestMethod.POST,
	consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }, produces
	= {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
	@ResponseStatus(HttpStatus.CREATED)
	@ResponseBody
	public StatusBean create1( @RequestBody Employee emp ) {
		/*
		 * Get all the request header information
		 * Request will be http://localhost:8080/restspring/emp/create1
		 */
		System.out.println("-------- Inside Method create1 with no HttpServletRequest
Object -------");
		ServletRequestAttributes attr = (ServletRequestAttributes)
RequestContextHolder.currentRequestAttributes();
		HttpServletRequest request = attr.getRequest();
		Enumeration en = request.getHeaderNames();
		while( en.hasMoreElements() ) {
			Object obj = en.nextElement();
			System.out.println("Header Name : "+obj);
			System.out.println("Header Value : "+request.getHeader((String)obj));
		}

		StatusBean statusBean = new StatusBean();
		if( empList.contains(emp) )
		{
			statusBean.setErrCode("001");
			statusBean.setErrMsg("Employee with this id "+emp.getId()+" already exists");
		}
		else
		{
			empList.add(emp);
			statusBean.setStatus("Success");
		}
		return statusBean;
	}

	@RequestMapping(value = "create2", method = RequestMethod.POST,
	consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }, produces
	= {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
	@ResponseStatus(HttpStatus.CREATED)
	@ResponseBody
	public StatusBean create2( @RequestBody Employee emp , HttpServletRequest request) {
		/*
		 * Get all the request header information
		 * Request will be http://localhost:8080/restspring/emp/create2
		 */
		System.out.println("-------- Inside Method create2 with HttpServletRequest Object
as method parameter -------");
		Enumeration en = request.getHeaderNames();
		while( en.hasMoreElements() ) {
			Object obj = en.nextElement();
			System.out.println("Header Name : "+obj);
			System.out.println("Header Value : "+request.getHeader((String)obj));
		}

		StatusBean statusBean = new StatusBean();
		if( empList.contains(emp) )
		{
			statusBean.setErrCode("001");
			statusBean.setErrMsg("Employee with this id "+emp.getId()+" already exists");
		}
		else
		{
			empList.add(emp);
```

```
			statusBean.setStatus("Success");
		}
		return statusBean;
	}

	@RequestMapping(value = "create3", method = RequestMethod.POST,
	consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }, produces
	= {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
	@ResponseStatus(HttpStatus.CREATED)
	@ResponseBody
	public StatusBean create3( @RequestBody Employee emp , HttpServletRequest request ,
HttpServletResponse response ) {
		/*
		 * Get all the request header information
		 * Request will be http://localhost:8080/restspring/emp/create3
		 */
		System.out.println("-------- Inside Method create3 with HttpServletResponse Object
as method parameter -------");
		System.out.println("Response Object : "+response);
		Enumeration en = request.getHeaderNames();
		while( en.hasMoreElements() ) {
			Object obj = en.nextElement();
			System.out.println("Header Name : "+obj);
			System.out.println("Header Value : "+request.getHeader((String)obj));
		}

		StatusBean statusBean = new StatusBean();
		if( empList.contains(emp) )
		{
			statusBean.setErrCode("001");
			statusBean.setErrMsg("Employee with this id "+emp.getId()+" already exists");
		}
		else
		{
			empList.add(emp);
			statusBean.setStatus("Success");
		}

		try {
			//Send a response error message
			response.sendError(HttpServletResponse.SC_SERVICE_UNAVAILABLE,"This service
is not available for you");
		} catch (IOException e) {
			e.printStackTrace();
		}

		return statusBean;
	}


	@RequestMapping(value = "update", method = RequestMethod.PUT,
	consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }, produces
	= {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
	@ResponseStatus( HttpStatus.CREATED )
	@ResponseBody
	public StatusBean update( @RequestBody Employee emp ) {
		StatusBean statusBean = new StatusBean();

		if( empList.contains(emp))
		{
			for( Employee emp1 : empList )
			{
				if( emp.equals(emp1))
				{
					empList.remove(emp1);
```

```
                              empList.add(emp);
                              statusBean.setStatus("Success");
                              break;
                        }
                  }
            }
            else {
                  statusBean.setErrCode("002");
                  statusBean.setErrMsg("Employee with this id "+emp.getId()+" does not exist");
            }
            return statusBean;
      }

      @RequestMapping(value = "delete", method = RequestMethod.DELETE,
      consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }, produces
      = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
      @ResponseStatus( HttpStatus.CREATED )
      @ResponseBody
      public StatusBean delete( @RequestBody Employee emp ) {
            StatusBean statusBean = new StatusBean();

            if( empList.contains(emp))
            {
                  for( Employee emp1 : empList )
                  {
                        if( emp.equals(emp1))
                        {
                              empList.remove(emp1);
                              statusBean.setStatus("Success");
                              break;
                        }
                  }
            }
            else {
                  statusBean.setErrCode("003");
                  statusBean.setErrMsg("Employee with this id "+emp.getId()+" does not exist,
hence data cannot be deleted");
            }
            return statusBean;
      }

      public void display() {
            for( Employee emps : empList )
                  System.out.println(emps);
            System.out.println("-----------------***************---------------------");
      }
}
```

WEB-INF/mvc-dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

      <context:component-scan base-package="com.ddlab.rnd.spring.rest" />
      <mvc:annotation-driven />

</beans>
```

web.xml

```
<web-app id="WebApp_ID" version="2.4"
	xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
	http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

	<display-name>Spring Web MVC Application</display-name>

	<servlet>
		<servlet-name>mvc-dispatcher</servlet-name>
		<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>

	<servlet-mapping>
		<servlet-name>mvc-dispatcher</servlet-name>
		<url-pattern>/*</url-pattern>
	</servlet-mapping>

	<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
	</context-param>

	<listener>
		<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
	</listener>

</web-app>
```

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>restspring</groupId>
	<artifactId>restspring</artifactId>
	<packaging>war</packaging>
	<version>0.0.1-SNAPSHOT</version>
	<name>restspring Maven Webapp</name>
	<url>http://maven.apache.org</url>

	<properties>
		<spring.version>3.2.2.RELEASE</spring.version>
		<jackson.version>1.9.10</jackson.version>
		<jdk.version>1.6</jdk.version>
	</properties>

	<dependencies>

		<!-- Spring 3 dependencies -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>${spring.version}</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-web</artifactId>
			<version>${spring.version}</version>
		</dependency>

		<dependency>
```

```
                <groupId>org.springframework</groupId>
                <artifactId>spring-webmvc</artifactId>
                <version>${spring.version}</version>
            </dependency>

            <!-- Jackson JSON Mapper -->
            <dependency>
                <groupId>org.codehaus.jackson</groupId>
                <artifactId>jackson-mapper-asl</artifactId>
                <version>${jackson.version}</version>
            </dependency>

            <!-- Servlet -API -->
            <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>servlet-api</artifactId>
                <version>2.5</version>
                <!-- <scope>provided</scope> -->
            </dependency>
        </dependencies>

        <build>
            <finalName>restspring</finalName>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>2.3.2</version>
                    <configuration>
                        <source>${jdk.version}</source>
                        <target>${jdk.version}</target>
                    </configuration>
                </plugin>
            </plugins>
        </build>

</project>
```

Testing

GET Request as XML

```
http://localhost:8080/restspring/emp/ID/Id1
In the header, provide
```

Accept : Application/xml

Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Status>
    <errCode>002</errCode>
    <errMsg>Employee with this id Id1 does not exist</errMsg>
    <status>Failed</status>
</Status>
```

GET Request as JSON

```
http://localhost:8080/restspring/emp/ID/Id1
In the header, provide
```

Accept : Application/json

Output

```
{
```

```
    "status": "Failed",
    "errCode": "002",
    "errMsg": "Employee with this id Id1 does not exist"
}
```

POST Request as XML

http://localhost:8080/restspring/emp/create

Content-type : application/xml
Accept : application/xml

XML Request

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
    <act>
        <actNo>11111</actNo>
        <name>DD Mishra</name>
    </act>
    <adrs>
        <countryName>India</countryName>
        <streetName>Bangalore</streetName>
    </adrs>
    <id>Id1</id>
    <name>Name-1</name>
</employee>
```

Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Status>
    <errCode></errCode>
    <errMsg></errMsg>
    <status>Success</status>
</Status>
```

POST Request as JSON

http://localhost:8080/restspring/emp/create

Content-type : application/json
Accept : application/json

JSON Request

```
{
  "Id" : "Id5",
  "Name" : "Name-1",
  "Adrs" : {
    "StreetName" : "Bangalore",
    "CountryName" : "India"
  },
  "Account" : {
    "AccountName" : "DD Mishra",
    "AccountNo" : "11111"
  }
}
```

Output

```
{
    "status": "Success",
    "errCode": "",
    "errMsg": ""
}
```

PUT Request as XML

http://localhost:8080/springrest1/emp/update

Content-type : application/json

Accept : application/json

XML Request

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
    <act>
        <actNo>11111</actNo>
        <name>Deba Mishra</name>
    </act>
    <adrs>
        <countryName>India</countryName>
        <streetName>Bhubaneswar</streetName>
    </adrs>
    <id>Id1</id>
    <name>Name-1</name>
</employee>
```

Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Status>
    <errCode></errCode>
    <errMsg></errMsg>
    <status>Success</status>
</Status>
```

PUT Request as JSON

---

http://localhost:8080/springrest1/emp/update

Content-type : application/json

Accept : application/json

JSOn Request

```
{
  "Id" : "Id1",
  "Name" : "Debadatta Mishra",
  "Adrs" : {
    "StreetName" : "Odisha",
    "CountryName" : "India"
  },
  "Account" : {
    "AccountName" : "Debadatta Mishra",
    "AccountNo" : "11111"
  }
}
```

Output

```
{
    "status": "Success",
    "errCode": "",
    "errMsg": ""
}
```

Delete Request as XML
http://localhost:8080/springrest1/emp/delete

Content-Type : application/xml
Accept : application/xml

XML Request
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
    <act>
        <actNo>11111</actNo>
        <name>Deba Mishra</name>
    </act>
    <adrs>
        <countryName>India</countryName>
        <streetName>Bangalore</streetName>
    </adrs>
    <id>Id1</id>
    <name>Name-1</name>
</employee>
```

Output
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Status>
    <errCode></errCode>
    <errMsg></errMsg>
    <status>Success</status>
</Status>
```

Delete Request as JSON
http://localhost:8080/springrest1/emp/delete

Content-Type : application/json
Accept : application/json

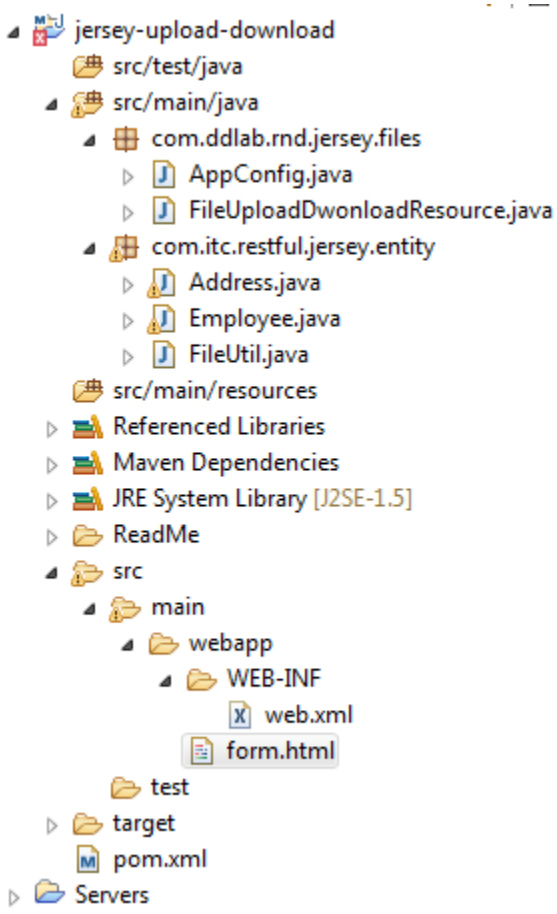
JSON Request
```
{
  "Id" : "Id2",
  "Name" : "Debadatta Mishra",
  "Adrs" : {
    "StreetName" : "Odisha",
    "CountryName" : "India"
  },
  "Account" : {
    "AccountName" : "Debadatta Mishra",
    "AccountNo" : "11111"
  }
}
```

Output
```
{
    "status": "Success",
    "errCode": "",
    "errMsg": ""
}
```

Difference between ACCEPT and Content-Type
ACCEPT is what the browser is able to digest
Content-Type is what format the actual data is in,
Content-Type is the format of the data to be sent to the server.
Accept is the format of the data sent by the server.

File Upload and Download using Jersey

Application Structure



Maven Configuration (pom.xml)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>jersey-upload-download</groupId>
	<artifactId>jersey-upload-download</artifactId>
	<packaging>war</packaging>
	<version>0.0.1-SNAPSHOT</version>
	<name>jersey-upload-download Maven Webapp</name>
	<url>http://maven.apache.org</url>

	<dependencies>

		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>3.8.1</version>
			<scope>test</scope>
		</dependency>
```

```
<!—Important Libraries for File Upload and Download -->
            <dependency>
                  <groupId>javax.ws.rs</groupId>
                  <artifactId>javax.ws.rs-api</artifactId>
                  <version>2.0.1</version>
            </dependency>

            <dependency>
                  <groupId>org.glassfish.jersey.core</groupId>
                  <artifactId>jersey-common</artifactId>
                  <version>2.14</version>
            </dependency>

            <dependency>
                  <groupId>org.glassfish.jersey.core</groupId>
                  <artifactId>jersey-server</artifactId>
                  <version>2.14</version>
            </dependency>
            <dependency>
                  <groupId>org.glassfish.jersey.core</groupId>
                  <artifactId>jersey-client</artifactId>
                  <version>2.14</version>
            </dependency>
            <dependency>
                  <groupId>org.glassfish.jersey.media</groupId>
                  <artifactId>jersey-media-multipart</artifactId>
                  <version>2.14</version>
            </dependency>
            <dependency>
                  <groupId>org.glassfish.jersey.containers</groupId>
                  <artifactId>jersey-container-servlet</artifactId>
                  <version>2.14</version>
            </dependency>

            <dependency>
                  <groupId>org.codehaus.jackson</groupId>
                  <artifactId>jackson-core-asl</artifactId>
                  <version>1.9.13</version>
            </dependency>
            <dependency>
                  <groupId>org.codehaus.jackson</groupId>
                  <artifactId>jackson-mapper-asl</artifactId>
                  <version>1.9.13</version>
            </dependency>
            <!-- The following is required for object to json -->
            <dependency>
                  <groupId>org.glassfish.jersey.media</groupId>
                  <artifactId>jersey-media-json-jackson</artifactId>
                  <version>2.14</version>
            </dependency>
      </dependencies>

      <build>
            <finalName>jersey-upload-download</finalName>
      </build>
</project>
```

Deployment Descriptor (web.xml)

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
              http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
       <display-name>jersey-upload-download</display-name>
       <servlet>
              <servlet-name>com.ddlab.rnd.jersey.files.AppConfig</servlet-name>
              <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
              <init-param>
                     <param-name>javax.ws.rs.Application</param-name>
                     <param-value>com.ddlab.rnd.jersey.files.AppConfig</param-value>
              </init-param>
              <load-on-startup>1</load-on-startup>
       </servlet>
       <servlet-mapping>
              <servlet-name>com.ddlab.rnd.jersey.files.AppConfig</servlet-name>
              <url-pattern>/rest/*</url-pattern>
       </servlet-mapping>
       <welcome-file-list>
              <welcome-file>form.html</welcome-file>
       </welcome-file-list>

</web-app>
```

Html file (form.html)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Form Page</title>
</head>
<body>
<h1>Upload a File</h1>

       <form action="rest/files/upload" method="post" enctype="multipart/form-data">

          <p>
              Select a file : <input type="file" name="file" size="45" />
          </p>

          <input type="submit" value="Upload It" />
       </form>

       <br>
       <strong>Multiple Files Upload</strong>
       <form action="rest/files/multiupload" method="post" enctype="multipart/form-data">

          <p>
              Select first file : <input type="file" name="file1" >
          </p>

          <p>
              Select second file : <input type="file" name="file2" >
          </p>

          <p>
```

```
            Select third file : <input type="file" name="file3" >
        </p>

        <input type="submit" value="Upload It" />
    </form>

</body>
</html>
```

Java Files

Entity Files

FileUtil.java

```
package com.itc.restful.jersey.entity;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class FileUtil {

	public static byte[] readContents(String filePath) {

		File file = new File(filePath);
		byte[] buffer = new byte[(int) file.length()];
		InputStream inStream = null;
		try {
			inStream = new FileInputStream(file);
			inStream.read(buffer);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			try {
				if (inStream != null)
					inStream.close();
			} catch (Exception e2) {
			}
		}

		return buffer;

	}

	public static void saveFile(InputStream uploadedInputStream, String serverLocation) {

		OutputStream outpuStream = null;
		try {
			int read = 0;
			byte[] bytes = new byte[1024];

			outpuStream = new FileOutputStream(new File(serverLocation));
			while ((read = uploadedInputStream.read(bytes)) != -1) {
				outpuStream.write(bytes, 0, read);
			}
```

```
		} catch (IOException e) {
			e.printStackTrace();
		}
		finally {
			try {
				if( outpuStream != null ) {
					outpuStream.flush();
					outpuStream.close();
				}
			} catch (Exception e2) {
			}
		}
	}

}
```

Employee.java

```
package com.itc.restful.jersey.entity;

import java.io.IOException;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.JsonFactory;
import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.annotate.JsonProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import com.fasterxml.jackson.core.JsonParser;

@XmlRootElement(name = "Emp")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee {

	@XmlElement(name = "Name")
	@JsonProperty("Name")
	private String name;

	@XmlElement(name = "Age")
	@JsonProperty("Age")
	private int age;

	@XmlElement(name = "Email")
	@JsonProperty("Email")
	private String emailId;

	@XmlElement(name = "Adrs")
	@JsonProperty("Adrs")
	private Address adrs;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
```

```
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getEmailId() {
		return emailId;
	}

	public void setEmailId(String emailId) {
		this.emailId = emailId;
	}

	public Address getAdrs() {
		return adrs;
	}

	public void setAdrs(Address adrs) {
		this.adrs = adrs;
	}

	public String toJSON() {
		ObjectMapper mapper = new ObjectMapper();
		try {
			return mapper.writeValueAsString(this);
		} catch (JsonGenerationException e) {
			e.printStackTrace();
		} catch (JsonMappingException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return null;
	}

	public static Employee toEmp(String jsonString) {

		ObjectMapper mapper = new ObjectMapper();
		Employee emp = null;
		try {
			emp = (Employee) mapper.readValue(jsonString, Employee.class);
		} catch (JsonParseException e) {
			e.printStackTrace();
		} catch (JsonMappingException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return emp;
	}
}
```

Address.java

```
package com.itc.restful.jersey.entity;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import org.codehaus.jackson.annotate.JsonProperty;
```

```java
@XmlRootElement(name = "Adrs")
@XmlAccessorType(XmlAccessType.FIELD)
public class Address {

	@XmlElement(name = "DoorNo")
	@JsonProperty("DoorNo")
	private String doorNo;

	@XmlElement(name = "Street")
	@JsonProperty("Street")
	private String streetId;

	@XmlElement(name = "City")
	@JsonProperty("City")
	private String city;

	@XmlElement(name = "Country")
	@JsonProperty("Country")
	private String country;

	public String getDoorNo() {
		return doorNo;
	}

	public void setDoorNo(String doorNo) {
		this.doorNo = doorNo;
	}

	public String getStreetId() {
		return streetId;
	}

	public void setStreetId(String streetId) {
		this.streetId = streetId;
	}

	public String getCity() {
		return city;
	}

	public void setCity(String city) {
		this.city = city;
	}

	public String getCountry() {
		return country;
	}

	public void setCountry(String country) {
		this.country = country;
	}
}
```

Resource Files

AppConfig.java

```
package com.ddlab.rnd.jersey.files;

import org.glassfish.jersey.jackson.JacksonFeature;
import org.glassfish.jersey.media.multipart.MultiPartFeature;
import org.glassfish.jersey.server.ResourceConfig;

public class AppConfig extends ResourceConfig {

	public AppConfig() {

		//For MultiPart, you have to register MultiPartFeature.class
		super(FileUploadDwonloadResource.class, JacksonFeature.class, MultiPartFeature.class);

	}

}
```

FileUploadDownloadResource.java

```
package com.ddlab.rnd.jersey.files;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import java.util.Map;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

import org.glassfish.jersey.media.multipart.BodyPart;
import org.glassfish.jersey.media.multipart.BodyPartEntity;
import org.glassfish.jersey.media.multipart.ContentDisposition;
import org.glassfish.jersey.media.multipart.FormDataBodyPart;
import org.glassfish.jersey.media.multipart.FormDataContentDisposition;
import org.glassfish.jersey.media.multipart.FormDataMultiPart;
import org.glassfish.jersey.media.multipart.FormDataParam;

import com.itc.restful.jersey.entity.Employee;
import com.itc.restful.jersey.entity.FileUtil;

@Path("/files")
public class FileUploadDwonloadResource {

	public static final String SERVER_UPLOAD_LOCATION_FOLDER = "D:/temp/";

	@POST
	@Path("/upload1")
	@Consumes(MediaType.MULTIPART_FORM_DATA)
	public Response uploadFile(FormDataMultiPart form) {
		// http://localhost:8080/jersey-upload-download/rest/files/upload1
		FormDataBodyPart filePart = form.getField("file");
		ContentDisposition headerOfFilePart = filePart.getContentDisposition();
		InputStream fileInputStream = filePart.getValueAs(InputStream.class);
		String filePath = SERVER_UPLOAD_LOCATION_FOLDER
				+ headerOfFilePart.getFileName();
		// save the file to the server
		FileUtil.saveFile(fileInputStream, filePath);
		String output = "File saved to server location using FormDataMultiPart : "
				+ filePath;
		return Response.status(200).entity(output).build();
	}
```

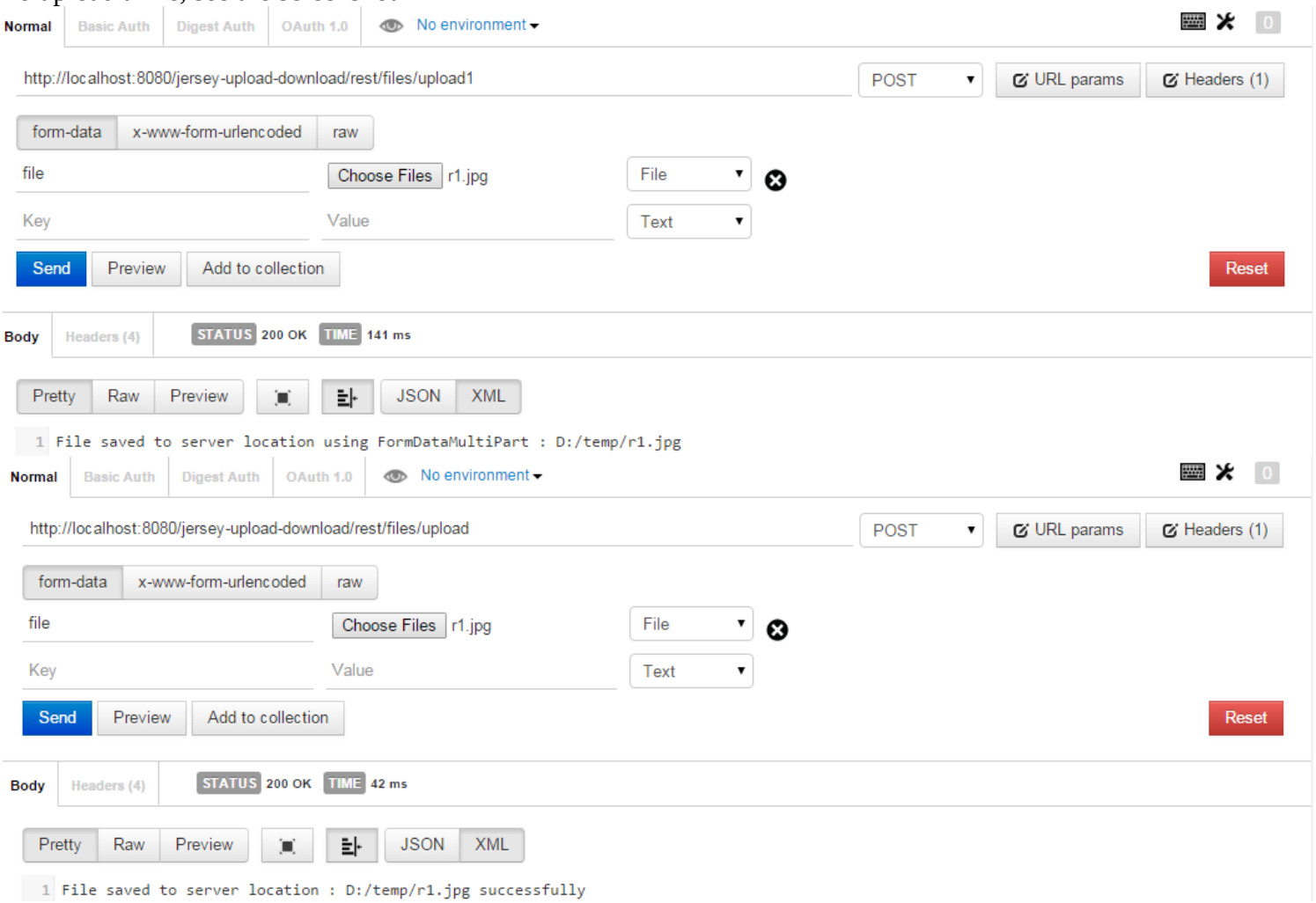
```
    // Another way of file uploading

    @POST
    @Path("/upload")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    public Response uploadFile(
                @FormDataParam("file") InputStream fileInputStream,
                @FormDataParam("file") FormDataContentDisposition contentDisposition)
{
          // http://localhost:8080/jersey-upload-download/rest/files/upload
          String filePath = SERVER_UPLOAD_LOCATION_FOLDER
                      + contentDisposition.getFileName();
          // save the file to the server
          String output = "";
          try {
                FileUtil.saveFile(fileInputStream, filePath);
                output = "File saved to server location : " + filePath
                            + " successfully";
          } catch (Exception e) {
                e.printStackTrace();
                output = "Unexpected server exception while uploading the file.";
          }
          return Response.status(200).entity(output).build();
    }

    @POST
    @Path("/upload2")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    public Response uploadFile(FormDataMultiPart form,
                @FormDataParam("emp") String emp) {

          // http://localhost:8080/jersey-upload-download/rest/files/upload2
          System.out.println("Employee-------->" + emp.toString());
          FormDataBodyPart filePart = form.getField("file");
          ContentDisposition headerOfFilePart = filePart.getContentDisposition();
          InputStream fileInputStream = filePart.getValueAs(InputStream.class);
          String filePath = SERVER_UPLOAD_LOCATION_FOLDER
                      + headerOfFilePart.getFileName();
          // save the file to the server
          FileUtil.saveFile(fileInputStream, filePath);
          String output = "File saved to server location using FormDataMultiPart : "
                      + filePath;
          return Response.status(200).entity(output).build();
    }
```

```
@POST
@Path("/upload3")
@Consumes(MediaType.MULTIPART_FORM_DATA)
public Response uploadFileWithData(FormDataMultiPart form) {
	// In this case, you extract the emp key from the
	// http://localhost:8080/jersey-upload-download/rest/files/upload3
	Map<String, List<FormDataBodyPart>> params = form.getFields();
	System.out.println(params);
	String myJson = params.get("emp").get(0).getValue();
	System.out.println("myjson---------->" + myJson);
	Employee emp = Employee.toEmp(myJson);
	System.out.println("emp-------->" + emp);

	FormDataBodyPart filePart = form.getField("file");
	ContentDisposition headerOfFilePart = filePart.getContentDisposition();
	InputStream fileInputStream = filePart.getValueAs(InputStream.class);
	String filePath = SERVER_UPLOAD_LOCATION_FOLDER
			+ headerOfFilePart.getFileName();
	// save the file to the server
	FileUtil.saveFile(fileInputStream, filePath);
	String output = "File saved to server location using FormDataMultiPart : "
			+ filePath;
	try {
		fileInputStream.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
	return Response.status(200).entity(output).build();
}

@POST
@Path("/multiupload")
@Consumes(MediaType.MULTIPART_FORM_DATA)
public Response uploadMultipleFile(FormDataMultiPart form) {
	// http://localhost:8080/jersey-upload-download/rest/files/multiupload

	BodyPartEntity bodyPartEntity;
	for (BodyPart part : form.getBodyParts()) {
		bodyPartEntity = (BodyPartEntity) part.getEntity();
		String fileName = part.getContentDisposition().getFileName();
		System.out
		.println("part.getContentDisposition().getFileName()---------->"
				+ fileName);
		InputStream fileInputStream = bodyPartEntity.getInputStream();
		String filePath = SERVER_UPLOAD_LOCATION_FOLDER + fileName;
		FileUtil.saveFile(fileInputStream, filePath);
	}
	return Response.status(200)
			.entity("All Files uploaded successfully ... ").build();
}
```

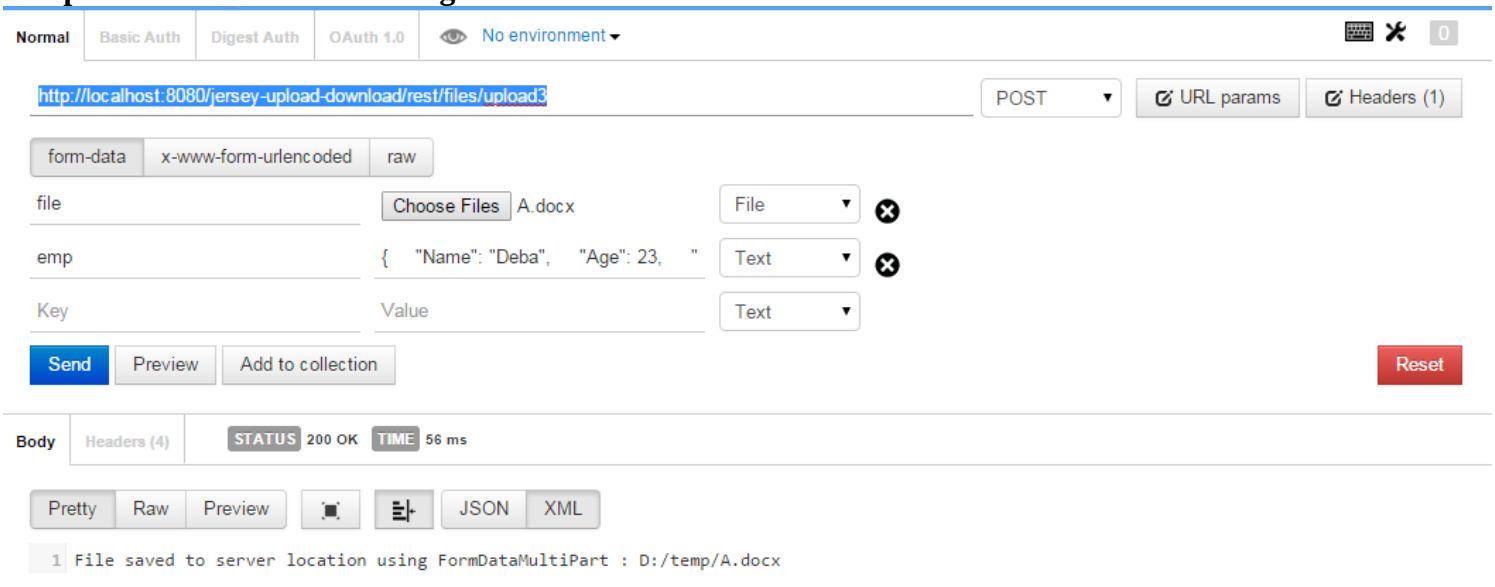
```
	@Path("/download")
	@GET
	@Produces(MediaType.APPLICATION_OCTET_STREAM)
	public Response getFile(@QueryParam("fileName") String fileName) {
		// Do not provide path param
		Response response = null;
		String dirPath = "D:/temp";
		String filePath = dirPath + File.separator + fileName;
		try {
			if (!new File(filePath).exists())
				throw new FileNotFoundException();
			byte[] docStream = FileUtil.readContents(dirPath + File.separator
					+ fileName);
			String attachment = "attachment; filename = " + fileName;
			response = Response
					.ok(docStream, MediaType.APPLICATION_OCTET_STREAM)
					.header("content-disposition", attachment).build();

		} catch (FileNotFoundException e) {
			response = Response.serverError().status(Status.BAD_REQUEST)
					.build();
		}
		return response;
	}

	@Path("/show/image")
	@GET
	@Produces("image/jpg")
	public Response showImage(@QueryParam("fileName") String fileName) {

		Response response = null;
		String dirPath = "D:/temp";
		InputStream inStream = null;
		String filePath = dirPath + File.separator + fileName;
		try {
			if (!new File(filePath).exists())
				throw new FileNotFoundException();
			inStream = new FileInputStream(filePath);
			response = Response.ok().entity(inStream).build();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			response = Response.serverError().status(Status.BAD_REQUEST)
					.build();
		}
		return response;
	}

}
```

How to test

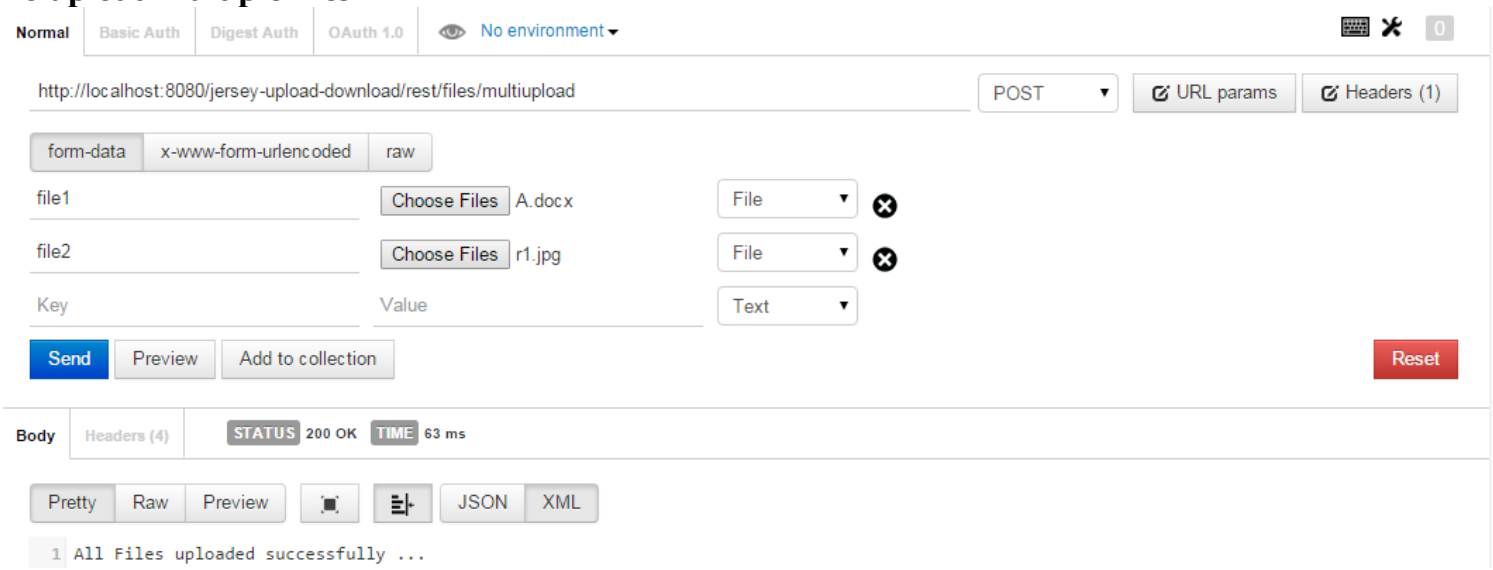
WADL URL : **http://localhost:8080/jersey-upload-download/rest/application.wadl**
Use only chrome postman client to test the above functionality
To upload a file, see the screenshot



To upload file with data as String



To upload multiple files



To download a file, use the following url and hit it in the browser. It does not need any rest client.
http://localhost:8080/jersey-upload-download/rest/files/show/image?fileName=r1.jpg
To show the image, use the following url and see the screenshot.
http://localhost:8080/jersey-upload-download/rest/files/show/image?fileName=r1.jpg

Important points

1. You can not use @PathParam for download a file. You have to use @QueryParam.

File Upload and Download using Spring REST API

Javacode "**FileUploadController.java**"

```
package com.ddlab.spring.rest;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
import org.springframework.util.FileCopyUtils;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;

@Controller
@RequestMapping({"/users"})
class FileUploadController
{
  @RequestMapping(value={"/photo"}, method={RequestMethod.POST})
  @ResponseBody
  String writeUserProfilePhoto(@RequestParam("file") MultipartFile file)
    throws Exception
  {
    System.out.println("File Name : "+file.getName());
    System.out.println("Original File Name : "+file.getOriginalFilename());

    String fileName = file.getOriginalFilename();
    String filePath = "D:/temp"+File.separator+fileName;
//    byte[] bytesForProfilePhoto = FileCopyUtils.copyToByteArray(file.getInputStream());
//    MediaType mt = MediaType.parseMediaType(file.getContentType());
    OutputStream out = new FileOutputStream(filePath);
    FileCopyUtils.copy(file.getInputStream(), out);
    HttpHeaders httpHeaders = new HttpHeaders();
    String responseStr = fileName+" has been uploaded successfully ...";

    return responseStr;
  }
}
```

mvc-dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

        <context:component-scan base-package="com.ddlab.spring.rest" />
        <mvc:annotation-driven />
```

```
        <bean id="multipartResolver"  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
                <property name="maxUploadSize" value="5000000"/>
        </bean>

</beans>
```

web.xml

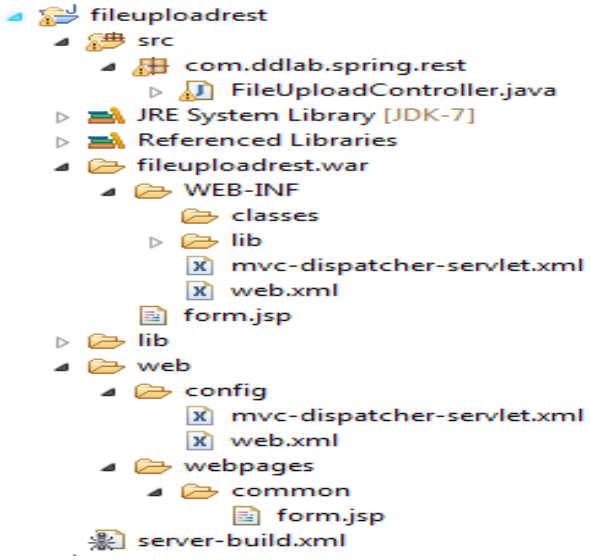
```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
                        version="3.0">

  <display-name>File Upload Using Spring REST</display-name>

  <servlet>
      <servlet-name>mvc-dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
      <servlet-name>mvc-dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
  </context-param>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <welcome-file-list>
         <welcome-file>form.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

form.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Form Page</title>
</head>
<body>
<h1>Upload a File</h1>

      <form action="users/photo" method="post" enctype="multipart/form-data">

         <p>
            Select a file : <input type="file" name="file" size="45" />
         </p>

         <input type="submit" value="Upload It" />
      </form>

</body>
</html>
```

List of jar files required

aopalliance-1.0.jar
commons-fileupload-1.3.1.jar
commons-io-2.4.jar
commons-logging-1.1.1.jar
servlet-api-2.5.jar
spring-aop-3.2.4.RELEASE.jar
spring-beans-3.2.4.RELEASE.jar
spring-context-3.2.4.RELEASE.jar
spring-core-3.2.4.RELEASE.jar
spring-expression-3.2.4.RELEASE.jar
spring-web-3.2.4.RELEASE.jar
spring-webmvc-3.2.4.RELEASE.jar



File Download using Spring REST API

Java code "**FileDownloadController.java**"

```
package com.ddlad.spring.rest;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.util.FileCopyUtils;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("/file")
public class FileDownloadController {

	public static String FILE_PATH = "D:"+File.separator+"myfile.txt";
	@RequestMapping(value = "/download", method = RequestMethod.GET)
	@ResponseBody
	public void download(HttpServletResponse response)  {
		try  {
			File file = new File(FILE_PATH);
			String attachment = "attachment; filename="+file.getName();
```

```
                response.setHeader("Content-Disposition", attachment);
                //response.setHeader("Content-Disposition", "attachment; filename=myfile.txt");
                InputStream inStream = new FileInputStream(file.getAbsolutePath());
                FileCopyUtils.copy(inStream, response.getOutputStream());
            }  catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            System.out.println("Hopefully file will be downloaded successfully .....");
        }
}
```

Spring configuration file "**mvc-dispatcher-servlet.xml**"

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context-3.0.xsd
         http://www.springframework.org/schema/mvc
         http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

       <context:component-scan base-package="com.ddlad.spring.rest" />
       <mvc:annotation-driven />
</beans>
```

Web.xml

```
<web-app id="WebApp_ID" version="2.4"
       xmlns="http://java.sun.com/xml/ns/j2ee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
       http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>File Download Using Spring REST</display-name>

  <servlet>
      <servlet-name>mvc-dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
      <servlet-name>mvc-dispatcher</servlet-name>
    <url-pattern>*.html</url-pattern>
  </servlet-mapping>
```

```
    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
  </context-param>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

Login.jsp

```
<!DOCTYPE html>
<html>
<head>
<title>File Download Using Spring REST</title>
</head>
<body>
      <h4>File Download Using Spring REST</h4>
      <a href="file/download.html">Download a file</a>
</body>
</html>
```

Complete REST URL to download : **http://localhost:8080/filedownloadrest/file/download.html**

List of Jar files required

aopalliance-1.0.jar , commons-logging-1.1.1.jar , servlet-api-2.5.jar , spring-aop-3.2.4.RELEASE.jar
spring-beans-3.2.4.RELEASE.jar
spring-context-3.2.4.RELEASE.jar
spring-core-3.2.4.RELEASE.jar
spring-expression-3.2.4.RELEASE.jar
spring-web-3.2.4.RELEASE.jar
spring-webmvc-3.2.4.RELEASE.jar

REST CALL URL FORMATION

```
@Path("itc")
public class ITCInfotechServices {

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getOrganisationName() {
		//GET http://localhost:8080/jersey-spring-rest/itc
		return "ITC Infotech, Bangalore, Karnataka";
	}

	@Path("/address")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddress() {
		// URL : http://localhost:8080/itcrestful/itc/address
		return "ITC Infotech India Limited, 18, Banaswadi Main Rd, Maruthi Sevanagar, Bangalore, 560005";
	}

	@Path("/address/{areaCode}")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddressByCode(@PathParam("areaCode") String areaCode) {
		// GET http://localhost:8080/jersey-spring-rest/itc/address/USA
	}

	@Path("/regionaladdress/{areaCode}")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getAddressByCountry(@PathParam("areaCode") String areaCode,
			@QueryParam("country") String country) {
		//GET http://localhost:8080/jersey-spring-rest/itc/regionaladdress/Europe?country=FI
	}

	@Path("/itcaddress")
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getITCAddress(@MatrixParam("country") String country,
			@MatrixParam("areacode") String areaCode) {
		// GET http://localhost:8080/jersey-spring-rest/itc/itcaddress;country=FI;areacode=europe
		return getAddressByCountry(areaCode, country);
	}

	@Path("/postaddress")
	@POST
	@Produces(MediaType.TEXT_PLAIN)
	public String postNGetITCAddress(@FormParam("country") String country,
			@FormParam("areacode") String areaCode) {
		// http://localhost:8080/itcrestful/itc/postaddress
		return getAddressByCountry(areaCode, country);
	}
```

In case of Firefox Rest Client
Set in the header
“name” = “Content-Type” and “value” = “application/x-www-form-urlencoded”
Then set the header as
country - FI
areacode - europe

In case of Chrome Postman
Click on x-www-form-urlencoded in Postman
country - FI
areacode - europe

```
@Path("/emp/{id}")
@GET
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public Response getEmployee(@PathParam("id") String id) {

GET http://localhost:8080/itcrestful/itc/emp/id123

}

@Path("/createemp")
@POST
@Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public Response createEmp(Employee emp) {

}
```

Emp creation
POST http://localhost:8080/jersey-spring-rest/itc/createemp
For Emp creation

```
<Emp>
   <Name>Deba</Name>
   <Age>23</Age>
   <Email>deba@gmail.com</Email>
   <Adrs>
      <DoorNo>12-A</DoorNo>
      <Street>Street-11</Street>
      <City>Bangalore</City>
      <Country>Karnataka</Country>
   </Adrs>
</Emp>
```

or

```
{
   "Name": "Deba",
   "Age": 23,
   "Email": "deba@gmail.com",
   "Adrs": {
```

```
        "DoorNo": "12-A",
        "Street": "Street-11",
        "City": "Bangalore",
        "Country": "Karnataka"
    }
}
```

Accept - application/xml or application/json
Content-Type ---------- application/xml or application/json

```
        @POST
        @Path("/upload")
        @Consumes(MediaType.MULTIPART_FORM_DATA)
        public Response uploadFile(FormDataMultiPart form) {

        }

}
```

REST VS SOAP

https://msdn.microsoft.com/en-US/magazine/Dd942839.aspx

Which is better, REST or SOAP?

Both REST and SOAP are often termed "Web services,"

REST is an architectural style for building client-server applications. SOAP is a protocol specification for exchanging data between two endpoints.

because REST relies on the semantics of HTTP, requests for data (GET requests) can be cached. RPC systems generally have no such infrastructure (and even when performing RPC using SOAP over HTTP, SOAP responses can't be cached because SOAP uses the HTTP POST verb, which is considered unsafe).

What about security? Isn't SOAP more secure than REST?

secure conversations over HTTP is now called Transport Layer Security (TLS), SSL is still the name most commonly used. What is true is that a SOAP-based service, because of the extra protocols specified in the various WS-* specifications, does support end-to-end message security. This means that if you pass SOAP messages from endpoint to endpoint to endpoint, over the same or different protocols, the message is secure. If your application needs this particular feature, SOAP plus WS-* is definitely the way to go.

What about transactions?

Here is another area in which SOAP and WS-* have explicit support for an "advanced" feature and REST has none. WS-Atomic Transactions supports distributed, two-phase commit transactional semantics over SOAP-based services. REST has no support for distributed transactions. You can have a resource called Transaction. When your client needs to do something transactional (such as transferring money between two bank accounts), the client creates a Transaction resource that specifies all the correct resources affected (in my example, the two bank accounts) by doing a POST to the Transaction factory URI. The client can then perform updates by sending a PUT to the transaction URI and close the transaction by sending a DELETE to the URI.

This, of course, requires some amount of hand-coding and explicit control over your system, whereas the WS-Atomic Transactions system is more automatic

If your system absolutely needs atomic transactional semantics across diverse systems, WS-Atomic Transactions is probably the way to go.

What about interoperability? Isn't SOAP supposed to be about interoperability? Isn't SOAP more interoperable than REST?

If you define interoperability as the technical ability to communicate between two divergent endpoints, I assert that REST wins the interoperability battle hands down.

Since one of the driving points behind creating the SOAP specification was to create an interoperable way to communicate between different platforms and different languages, many people are surprised by this assertion. But a funny thing happened on the way to widespread interoperability: the WS-* specifications (and vendors' implementations of said specifications) made SOAP services less interoperable rather than more interoperable.

The problem in the SOAP and WS-* arena is the large number of different standards (and versions of each of those standards) to choose from. And when a particular vendor chooses to implement a particular standard, that vendor often provides an implementation that is just slightly different from another vendor's (or all others). This leads to problems whenever you have to cross vendor boundaries (languages and operating system).
Of course, even to use SOAP you need a SOAP toolkit on your platform, which most (but not all) platforms have today. And then you have to deal with myriad WS-* specifications and figure out which to use (or not to use) and how that affects interoperability. To be honest, it's kind of a mess out there.
In terms of platforms, REST has the advantage because all you need to use REST is an HTTP stack (either on the client or the server).

But what about metadata? So what if REST is so interoperable—there's no WSDL with REST, and without WSDL, I can't generate a client-side proxy to call a service. REST is hard to use.

It's true that in the world of REST, there is no direct support for generating a client from server-side-generated metadata, as there is in the world of SOAP with Web Service Description Language (WSDL). A couple of efforts are being made to get such support into REST, one being a parallel specification, known as WADL (Web Application Description Language). The other is a push to use WSDL 2.0 to describe RESTful endpoints. I often say that REST is simple, but simple doesn't always mean easy. SOAP is easy (because of WSDL), but easy doesn't always mean simple.

SOAP services are much harder to scale than RESTful services, which is, of course, one of the reasons that REST is often chosen as the architecture for services that are exposed via the Internet (like Facebook, MySpace, Twitter, and so on).

Securing Your API – What To Do

http://blog.smartbear.com/security/api-security-in-rest-vs-soap/

Over the years, SOAP has added extensions to deal with transactional messaging specific security considerations. SOAP has been around long enough and has been adopted by large enterprises such that it benefits from OASIS and W3C recommendations. Mainly, XML-Encryption, XML-Signature, and SAML tokens help to tighten up the security story over the data being received by and sent from a SOAP service.

REST on the other hand does not implement any specific security patterns, mainly because the pattern focuses on how to deliver and consume data, not how to build in safety into the way you exchange data. Proper amounts of security in code, deployment, and transmission should be determined by those implementing REST architecture patterns, not presumed as something that comes out-of-box.

REST Vs SOAP, The Difference Between Soap And Rest

http://spf13.com/post/soap-vs-rest

SOAP

SOAP brings it's own protocol and focuses on exposing pieces of application logic (not data) as services. SOAP exposes operations. SOAP is focused on accessing named operations, each implement some business logic through different interfaces.

Why REST?

Since REST uses standard HTTP it is much simpler in just about ever way. Creating clients, developing APIs, the documentation is much easier to understand.
REST permits many different data formats where as SOAP only permits XML.
REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached.

Why SOAP?

WS-AtomicTransaction

Need ACID Transactions over a service, you're going to need SOAP. While REST supports transactions, it isn't as comprehensive and isn't ACID compliant. Fortunately ACID transactions almost never make sense over the internet. REST is limited by HTTP itself which can't provide two-phase commit across distributed transactional resources, but SOAP can. Internet apps generally don't need this level of transactional reliability, enterprise apps sometimes do.

WS-ReliableMessaging

Rest doesn't have a standard messaging system and expects clients to deal with communication failures by retrying. SOAP has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries.

REST vs SOAP: When Is REST Better?

The Argument For SOAP

SOAP is still offered by some very prominent tech companies for their APIs (Salesforce, Paypal, Docusign). One of the main reasons: legacy system support. If you built a connector between your application and Salesforce back in the day, there's a decent probability that connection was built in SOAP.

There are a few additional situations:

- SOAP is good for applications that require *formal contracts* between the API and consumer, since it can enforce the use of formal contracts by using WSDL (Web Services Description Language).
- Additionally, SOAP has built in *WS-Reliable messaging* to increase security in asynchronous execution and processing.
- Finally, SOAP has built-in *stateful operations*. REST is naturally stateless, but SOAP is designed support conversational state management.

Some would argue that because of these features, as well as support for WS_AtomicTransaction and WS_Security, SOAP can benefit developers when there is a high need for transactional reliability.

It is important to note that one of the advantages of SOAP is the use of the "generic" transport.

Understanding RESTful vs SOAP web services

http://funonrails.com/2014/03/understanding-restful-vs-soap-web-services/

REST

ighlights:

1. It is simple
2. It is lightweight and faster
3. Platform-independent
4. Language-independent
5. runs over HTTP
6. Response can be returned in "XML/JSON" format

Like Web Services, REST offers no built-in security features, encryption, session management, QoS guarantees, etc. But also as with Web Services, these can be added by building on top of HTTP:

SOAP

It defines 3 fundamental properties:

1. What service does: Operations provides
2. How service is accessed: Data format and protocol details
3. Where a service is located: Adress (URL) details

Highlights

Here are few **reasons** for which you may want to use SOAP:
Platform and laguage independent
Follows W3C standard
WSDL (Web services description language) is a W3C standard based on xml and is used to describe web services.
Security
It supports SSL (just like REST). It also provides a standard implementation of data integrity and data privacy. Being "Enterprise" it's more secure.
Atomic Transactions
It supports ACID Transactions (WS- Atomic Transactions) so it can provide two phase commit over distributed transactional resources.
Reliable Messaging
It has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries

WS-Security

https://en.wikipedia.org/wiki/WS-Security

Web Services Security (**WS-Security**, **WSS**) is an extension to SOAP to apply security to Web services. It is a member of the Web service specifications and was published by OASIS.

The protocol specifies how integrity and confidentiality can be enforced on messages and allows the communication of various security token formats, such as Security Assertion Markup Language (SAML), Kerberos, and X.509. Its main focus is the use of XML Signature and XML Encryption to provide end-to-end security.

Features[edit]

WS-Security describes three main mechanisms:

- How to sign SOAP messages to assure integrity. Signed messages also provide non-repudiation.
- How to encrypt SOAP messages to assure confidentiality.
- How to attach security tokens to ascertain the sender's identity.

The specification allows a variety of signature formats, encryption algorithms and multiple trust domains, and is open to various security token models, such as:

- X.509 certificates,
- Kerberos tickets,
- User ID/Password credentials,

- SAML Assertions, and
- custom-defined tokens.

The token formats and semantics are defined in the associated profile documents.

WS-Security incorporates security features in the header of a SOAP message, working in the application layer.

These mechanisms by themselves do not provide a complete security solution for Web services. Instead, this specification is a building block that can be used in conjunction with other Web service extensions and higher-level application-specific protocols to accommodate a wide variety of security models and security technologies. In general, WSS by itself does not provide any guarantee of security. When implementing and using the framework and syntax, it is up to the implementor to ensure that the result is not vulnerable.

Key management, trust bootstrapping, federation and agreement on the technical details (ciphers, formats, algorithms) is outside the scope of WS-Security.

What is non-repudiation

http://security.stackexchange.com/questions/6730/what-is-the-difference-between-authenticity-and-non-repudiation

Authentication and non-repudiation are two different sorts of concepts.

- Authentication is a **technical concept**: e.g., it can be solved through cryptography.
- Non-repudiation is a **legal concept**: e.g., it can only be solved through legal and social processes (possibly aided by technology).

Authenticity is about one party (say, Alice) interacting with another (Bob) to convince *Bob* that some data really comes from Alice.
Non-repudiation is about Alice showing to Bob a proof that some data really comes from Alice, such that not only Bob is convinced, but Bob also gets the assurance that he could show the same proof to Charlie, and Charlie would be convinced, too, even if Charlie does not trust Bob.

Therefore, a protocol which provides non-repudiation necessarily provides authenticity as a byproduct; in a way, authenticity is a sub-concept of non-repudiation. However, there are ways to provide authenticity (only) which are vastly more efficient than known methods to achieve signatures (authenticity can be obtained with a Message Authentication Code whereas non-repudiation requires aDigital Signature with much more involved mathematics). For this reason, it makes sense to use "authenticity" as a separate concept.
SSL/TLS is a tunneling protocol which provides authenticity (the client is sure to talk to the intended server) but not non-repudiation (the client cannot record the session and show it as proof, in case of a legal dispute with the server, because it would be easy to build a totally fake session record)

Nonrepudiation – Dictionary meaning

Nonrepudiation is the assurance that someone cannot deny something. Typically, nonrepudiation refers to the ability to ensure that a party to a contract or a communication cannot deny the authenticity of their signature on a document or the sending of a message that they originated.

HTTP Status Messages

1xx: Information

Message:	**Description:**
100 Continue	The server has received the request headers, and the client should proceed to send the request body
101 Switching Protocols	The requester has asked the server to switch protocols
103 Checkpoint	Used in the resumable requests proposal to resume aborted PUT or POST requests

2xx: Successful

Message:	**Description:**
200 OK	The request is OK (this is the standard response for successful HTTP requests)
201 Created	The request has been fulfilled, and a new resource is created
202 Accepted	The request has been accepted for processing, but the processing has not been completed
203 Non-Authoritative Information	The request has been successfully processed, but is returning information that may be from

	another source
204 No Content	The request has been successfully processed, but is not returning any content
205 Reset Content	The request has been successfully processed, but is not returning any content, and requires that the requester reset the document view
206 Partial Content	The server is delivering only part of the resource due to a range header sent by the client

3xx: Redirection

Message:	**Description:**
300 Multiple Choices	A link list. The user can select a link and go to that location. Maximum five addresses
301 Moved Permanently	The requested page has moved to a new URL
302 Found	The requested page has moved temporarily to a new URL
303 See Other	The requested page can be found under a different URL
304 Not Modified	Indicates the requested page has not been modified since last requested

306 Switch Proxy	*No longer used*
307 Temporary Redirect	The requested page has moved temporarily to a new URL
308 Resume Incomplete	Used in the resumable requests proposal to resume aborted PUT or POST requests

4xx: Client Error

Message:	**Description:**
400 Bad Request	The request cannot be fulfilled due to bad syntax
401 Unauthorized	The request was a legal request, but the server is refusing to respond to it. For use when authentication is possible but has failed or not yet been provided
402 Payment Required	*Reserved for future use*
403 Forbidden	The request was a legal request, but the server is refusing to respond to it
404 Not Found	The requested page could not be found but may be available again in the future
405 Method Not Allowed	A request was made of a page using a request method not supported by that page

406 Not Acceptable	The server can only generate a response that is not accepted by the client
407 Proxy Authentication Required	The client must first authenticate itself with the proxy
408 Request Timeout	The server timed out waiting for the request
409 Conflict	The request could not be completed because of a conflict in the request
410 Gone	The requested page is no longer available
411 Length Required	The "Content-Length" is not defined. The server will not accept the request without it
412 Precondition Failed	The precondition given in the request evaluated to false by the server
413 Request Entity Too Large	The server will not accept the request, because the request entity is too large
414 Request-URI Too Long	The server will not accept the request, because the URL is too long. Occurs when you convert a POST request to a GET request with a long query information
415 Unsupported Media Type	The server will not accept the request, because the media type is not supported
416 Requested Range	The client has asked for a portion of the file, but the server cannot

Not Satisfiable	supply that portion
417 Expectation Failed	The server cannot meet the requirements of the Expect request-header field

5xx: Server Error

Message:	**Description:**
500 Internal Server Error	A generic error message, given when no more specific message is suitable
501 Not Implemented	The server either does not recognize the request method, or it lacks the ability to fulfill the request
502 Bad Gateway	The server was acting as a gateway or proxy and received an invalid response from the upstream server
503 Service Unavailable	The server is currently unavailable (overloaded or down)
504 Gateway Timeout	The server was acting as a gateway or proxy and did not receive a timely response from the upstream server
505 HTTP Version Not Supported	The server does not support the HTTP protocol version used in the request
511 Network Authentication Required	The client needs to authenticate to gain network access