WSRF based HelloWorld with multiple instances

Version: 1.2 Date 20/03/06

This tutorial is the continuation of first and second tutorial "Converting a HelloWorld Web service into HelloWorld WSRF" and "Implementing OO based state-full HelloWorld" which can be found at http://tyne.dl.ac.uk/WOSE/Tutorial1.pdf and http://tyne.dl.ac.uk/WOSE/Tutorial2.pdf. In last two tutorials we had very simple "WSRF based HelloWorld" example, in both cases no matter how many clients are accessing our "WSRF based HelloWorld" there is only one copy of the Resource shared between them. In this tutorial we will change our "WSRF based HelloWorld" so that for each client there will be separate instance of the Resource, when ever a new instance will be created unique identifier will be sent to client along with the location of instance. Still two clients can interact with single instance, if they both use same unique identifier. Most of our deployment descriptors from previous tutorial are re-useable, but I have changed name of Web Service and target namespace from HelloWorldService2 to HelloWorldService3 and *http://tutorial2.wsrf.dl.ac.uk/helloworld to http://tutorial3.wsrf.dl.ac.uk/helloworld*. In fact based on each tutorial we will keep on modifying the deployment descriptors and configuration files referencing related tutorial number. These are although minor changes but can make difference later when we have to use stubs and skeleton generated by the Globus. If you remember from the previous tutorials WSDL2Java uses target namespace to generate package statement for stubs and skeletons unless we provide namespace to package mapping in optional file "NStoPkg.properties". We will still not use ns2mapping.properties in this tutorial, but later we will investigate the utility of this mapping feature. If you are interested and keen right now to see the working of "NStoPkg.properties" then better check the tutorial by Borja Sotomayor in "The Globus Toolkit 4 Programmer's Tutorial". In this tutorial we are using target namespace *http://tutorial3.wsrf.dl.ac.uk/helloworld* therefore stubs and skeletons created by WSDL2Java will be in the package *"uk.ac.dl.wsrf.tutorial3.helloworld.*"*, *"uk.ac.dl.wsrf.tutorial3.helloworld.service.*"* and *"uk.ac.dl.wsrf.tutorial3.helloworld.bindings.*"*, it is very important to know default namespace to package conversion as later we have to use classes in our Web Service implementation and Client implementation even when those classes are not existing. More on this, later when, we will start implementation. Last tutorial was to convert HelloWorld Web Service into WSRF based HelloWorld, this tutorial is about to convert WSRF based HelloWorld with single Resource Instance into WSRF based HelloWorld with multiple instances of Resource, the technique used here is convenient way but not recommended way. In next tutorial we will implement the same tutorial using Factory Design Pattern.

HelloWorld WSRF with multiple instances

In first tutorials we started with simple WSRF *Service*, which, you have already seen and discussed the changes which you need to make in the each of the following tutorials. I believe it is easy to understand and visualise when differences and similarities are represented in tabular way, rather than on different pages. Our simple WSRF *Service*, had two business methods, echo() and *getNameRP(),* and one compulsory method *GetResourceProperty()*. We will add new business method called *create(),* both in the WSDL file and in java implementation *HelloWorld.java.* Client can't interact with any business method of our web service unless first he calls the *create(),* in return create method will create new instance of resource used by our service and will return the unique identifier of the resource/service to the client. Client will use this unique identifier for further interaction with the web service. Unique identifier is in the form of EndPointReferenceType which not only contains the unique identifier of the resource object but also has the information of the URL of the service managing it. One Resource can be managed by different Services and one Service can manage different Resources therefore pair of unique identifier of the resource object and URL of the managing Service makes it complete EndPointReferenceType.

Modifying WSDL File:

Just like previous tutorials we are starting with our WSDL file, which is very similar to our last WSDL example with few following changes:

1. Change in target name space.

2. Adding new data types for create messages.
3. Adding new messages for create method.
4. Adding new create method in element "*portType*"

This WSDL file is in the directory "schema\tutorial" relative to tutorial "src" directory with the name "helloworld3_port_type.wsdl", in last tutorial our WSDL file was helloworld2_port_type.wsdl.

First change is all about changing target namespace, there are in total 4 changes highlighted by blue and bold font. Second change is to add data types related to create messages. There are two data types, one for input and other for output of the create method, create() doesn't takes any input, therefore it is empty. The output from create() method is of data type EndpointReferenceType. If you remember EndpointReferenceType is wrapper around URL of the service and the unique identifier.

```
<element name="Create">
        <complexType/>
</element>
<element name="CreateResponse" type="wsa:EndpointReferenceType" />
```

To use EndpointReferenceType we need following namespace:
xmlns:wsa=http://schemas.xmlsoap.org/ws/2004/03/addressing
and WS-Addressing.xsd which is imported in types element:
<import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
schemaLocation="../ws/addressing/WS-Addressing.xsd"/>
For convenience we have already imported all required WSDL and XMLSchema files so we don't need to add them for this tutorial, but it is better to know which WSDL and XMLSchema file is doing what.

Third change is to add messages related to create method based on data types added in last step, this is once again very basic stuff.

```
<message name="CreateRequest">
  <part name="CreateRequest" element="tns:Create" />
</message>
<message name="CreateResponse">
  <part name="CreateResponse" element="tns:CreateResponse" />
</message>
```

The final change in the WSDL file is to add create method in the element portType, be sure it is business method, it has nothing to do with imported WSDL and XMLSchema files.

```
<operation name="create">
    <input  name="CreateRequest" message="tns:CreateRequest" />
    <output name="CreateResponse" message="tns:CreateResponse" />
</operation>
```

The complete WSDL file helloworld3_port_type.wsdl is shown below with changes in different font:

<definitions name="HelloWorld"
*targetNamespace=**"http://tutorial3.wsrf.dl.ac.uk/helloworld"***
*xmlns:tns=**"http://tutorial3.wsrf.dl.ac.uk/helloworld"***
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"

```
  xmlns:gtwsdl1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
  xmlns:wsntw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
  xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
 <import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-01.wsdl"
  location="../wsrf/faults/WS-BaseFaults.wsdl"/>
 <import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
  location="../wsrf/lifetime/WS-ResourceLifetime.wsdl"/>
 <import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
  location="../wsrf/properties/WS-ResourceProperties.wsdl"/>
 <import namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
  location="../wsrf/notification/WS-BaseN.wsdl"/>
 <import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
  location="../wsrf/servicegroup/WS-ServiceGroup.wsdl"/>
 <import namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
  location="../wsrf/notification/WS-BaseN.wsdl" />

 <types>
  <schema
     targetNamespace="http://tutorial3.wsrf.dl.ac.uk/helloworld"
     xmlns:tns="http://tutorial3.wsrf.dl.ac.uk/helloworld"
     xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
     schemaLocation="../ws/addressing/WS-Addressing.xsd"/>
 <import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.xsd"
     schemaLocation="../wsrf/servicegroup/WS-ServiceGroup.xsd"/>

   <element name="name" type="xsd:string"/>

   <element name="getNameRPRequest" >
    <complexType/>
   </element>
   <element name="getNameRPResponse" type="xsd:string"/>

   <element name="HelloWorldResourcePropertiesSet">
    <complexType>
     <sequence>
      <element ref="tns:name"/>
     </sequence>
    </complexType>
   </element>

   <element name="echoRequest" type="xsd:string" />
   <element name="echoResponse" type="xsd:string" />

   <element name="Create">
      <complexType/>
   </element>
   <element name="CreateResponse" type="wsa:EndpointReferenceType" />
```

```
  </schema>
</types>

<message name="EchoRequest">
  <part name="EchoRequest"  element="tns:echoRequest" />
</message>
<message name="EchoResponse">
 <part name="EchoResponse" element="tns:echoResponse" />
</message>

  <message name="CreateRequest">
    <part name="CreateRequest" element="tns:Create" />
  </message>
  <message name="CreateResponse">
    <part name="CreateResponse" element="tns:CreateResponse" />
  </message>

<message name="GetNameRPRequest">
  <part name="GetNameRPRequest" element="tns: getNameRPRequest " />
</message>
<message name="GetNameRPResponse">
  <part name="GetNameRPResponse" element="tns:getNameRPResponse" />
</message>

<portType name="HelloWorldPortType"
      wsrp:ResourceProperties="HelloWorldResourcePropertiesSet">

  <operation name="GetResourceProperty">
   <input name="GetResourcePropertyRequest"
     message="wsrpw:GetResourcePropertyRequest"
     wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/GetResourceProperty"/>
   <output name="GetResourcePropertyResponse"
       message="wsrpw:GetResourcePropertyResponse"
       wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/GetResourcePropertyResponse"/>
   <fault  name="InvalidResourcePropertyQNameFault"
        message="wsrpw:InvalidResourcePropertyQNameFault"/>
    <fault name="ResourceUnknownFault" message="wsrpw:ResourceUnknownFault"/>
  </operation>

  <!--  name in input and output is optional-->
  <operation name="echo">
    <input  name="EchoRequest"  message="tns:EchoRequest" />
    <output name="EchoResponse" message="tns:EchoResponse" />
  </operation>

  <operation name="getNameRP">
    <input  name="GetNameRPRequest"  message="tns:GetNameRPRequest" />
    <output name="GetNameRPResponse" message="tns:GetNameRPResponse" />
  </operation>

 <operation name="create">
    <input  name="CreateRequest" message="tns:CreateRequest" />
    <output name="CreateResponse" message="tns:CreateResponse" />
</operation>
```

```
  </portType>
</definitions>
```

Modified Implementation of Web Service:

This implementation of *HelloQNames* is using the similar interface which we have used in last example and only change is in the NS variable which is changed from *http://tutorial2.wsrf.dl.ac.uk/helloworld* to *http://tutorial3.wsrf.dl.ac.uk/helloworld* .

```
public static final String NS = "http://tutorial3.wsrf.dl.ac.uk/helloworld";
```

This interface has few static and final variables. Interfaces can't be initialized and if interfaces have variables then they should be static and final, reason for static variables is very simple, as you can't create the object of interface therefore there is no way to access non-static variables and as there can be many objects using this interface and if anyone of them will change the value of static variable, change will be effective for all objects using this interface which may not be desired, thus keeping the variables final means that there is no possibility of even accidental change in value. In first tutorial where ever we had to use QName we have to create the instance of QName with target namespace URL and name of variable, which is cumbersome and error prone. This interface is just wrapping all QName used in our implementation. Below is the implementation of HelloQNames.java.

```
package uk.ac.dl.ws.service;

import javax.xml.namespace.QName;

public interface HelloQNames {

    public static final String NS = "http://tutorial3.wsrf.dl.ac.uk/helloworld";
    public static final QName RP_NAME = new QName(NS, "name");
    public static final QName RESOURCE_PROPERTIES = new QName(NS, "HelloWorldResourcePropertiesSet");

}
```

String NS is String value of our target namespace which is http://tutorial3.wsrf.dl.ac.uk/helloworld for tutorial 3. RP_NAME and RESOURCE_PROPERTIES are qualified name of our variable declared in our WSDL file. Only package we have to import is "*javax.xml.namespace.QName*" as this interface is utility interface and has nothing to do with WSRF implementation.

Now we start modifying HelloWorldResource.java which is implementation of Resource/s to declare and create resources, and in fact it is also initializing them, but no one will access it directly and all calls to initialize the tutorials should be through HelloWorldResourceHome.java. Below are the modifications in HelloWorldResource.java:

1. Importing two new class ResourceIdentifier and SimpleResourceProperty.
   ```
   import org.globus.wsrf.ResourceIdentifier;
   import org.globus.wsrf.impl.SimpleResourceProperty;
   ```
   SimpleResourceProperty is used instead of ReflectionResourceProperty.
2. HelloWorldResource.java will implement ResourceIdentifier also.
3. Private variable of type Object to hold unique ID.
   ```
   private Object id;
   ```

4. In last implementation ResourceProperty nameRP was declared and initialized in the initialize() method but this time it is made global variable outside any method.
```
/** Resource property "name". */ // making it global
private ResourceProperty nameRP;
```
5. In last tutorial nameRP was of type ReflectionResourceProperty.
```
nameRP = new ReflectionResourceProperty(HelloQNames.RP_NAME, "name", this);
```
But this time it is of type SimpleResourceProperty
```
this.nameRP = new SimpleResourceProperty(HelloQNames.RP_NAME);
```
6. In case of ReflectionResourceProperty variable name was added to nameRP at the time of initialization, which is not the case for SimpleResourceProperty, therefore we have to add variable "*name*" to the *nameRP* manually.
```
this.nameRP.add(name);
```
7. Another change is to add set method for *nameRP*
```
public void setNameRP(String nameValue) {
        setName(nameValue);
        this.nameRP.set(0, name);
}
```
8. Last change is to add get method for *ID*
```
public Object getID() {
        return id;
}
```

Below is the whole HelloWorldResource.java with modifications. Added source code is in blue font and the Source Code deleted from previous example is in green font.

```
package uk.ac.dl.ws.service;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.ResourceIdentifier;
import org.globus.wsrf.impl.SimpleResourceProperty;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;

public class HelloWorldResource  implements Resource, ResourceProperties, ResourceIdentifier {

  /** the identifier of this service */
  private Object id;

  /* Resource Property set */
  private ResourcePropertySet propSet;

  /* Resource properties */
  private String name;

  /** Resource property "name". */ // making it global
  private ResourceProperty nameRP;

  public void initialize() throws Exception {
```

```
    // choose an ID
    this.id = new Integer(hashCode());

    /* Create RP set */
    this.propSet = new SimpleResourcePropertySet(
        HelloQNames.RESOURCE_PROPERTIES);

    /* Initialize the RP's */
    try {
      ResourceProperty nameRP = new ReflectionResourceProperty(
          HelloQNames.RP_NAME, "name", this);

      // create resource properties
    this.nameRP = new SimpleResourceProperty(HelloQNames.RP_NAME);

      this.propSet.add(nameRP);
      setName("Asif Akram Tutorial 3");
      this.nameRP.add(name);

    }
    catch (Exception e) {
      throw new RuntimeException(e.getMessage());
    }
  }

  public String getName() {
    return name;
  }

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

  /* Required by interface ResourceProperties */
  public ResourcePropertySet getResourcePropertySet() {
    return this.propSet;
  }

  public void setNameRP(String nameValue) {
    setName(nameValue);
    this.nameRP.set(0, name);
  }

  public Object getID() {
    return id;
  }

}
```


We have one variable called “name” which will store the value passed to the echo method of our Web Service. This variable is Resource Property of our Web Service, this concept is very clearly and in detail explained by Borja Sotomayor in “The Globus Toolkit 4

Programmer's Tutorial" under the section Getting Started. Remember your WSRF Web Service can have many Resource Properties and all of them are placed in *ResourcePropertySet* for later retrieval, modification and deletion. It is always nice to follow good naming conventions which are self explanatory, data type which will encapsulate all Resources of our WSRF Web Services will ends with "*ResourcePropertiesSet"* to indicate that it is *ResourcePropertySet,* all individual resources will ends with RP (Resource Property) to keep things clear and simple.
We have initialized the *"propSet"* as object of *SimpleResourcePropertySet,* you may have figured out that *ResourcePropertySet* is interface and *SimpleResourcePropertySet* is concrete implementation of *ResourcePropertySet*. Constructor of *SimpleResourcePropertySet* takes QName i.e. the qualified name of our resource property as declared in the WSDL file. Similarly *ResourceProperty* is an interface and *ReflectionResourceProperty/* SimpleResourceProperty is one of its implementation used to declare and create variable *nameRP* (name Resource Property). ReflectionResourceProperty has three different constructors and the constructor used in the above example is: *ReflectionResourceProperty(QName name, String propertyName, Object obj)*, QName should match the qualified name in the WSDL and "String propertyName" is the name of variable in our implementation. It can be called anything like myName. SimpleResourceProperty(*QName name*) only takes QName and property has to be added later.
Then we have to add this newly created *ResourceProperty* into our *ResourcePropertySet*, *this.propSet.add(nameRP)* and then assign initial value by calling utility set method for private variable name.

Note, we are using our interface "HelloQNames.java" wherever we have to pass Qualified Name (Qname)

```
this.propSet = new SimpleResourcePropertySet(HelloQNames.RESOURCE_PROPERTIES);
nameRP = new SimpleResourceProperty(HelloQNames.RP_NAME);
this.nameRP.add(name);
```

Second class to be modified is "HelloWorldResourceHome.java", there is nothing common between "HelloWorldResourceHome.java" from previous tutorial (tutorial 2) and "HelloWorldResourceHome.java" from this tutorial (tutorial 3). If you remember "HelloWorldResourceHome.java" is only to initialize our "HelloWorldResource.java" but in last tutorial "HelloWorldResourceHome.java" can initialize only one instance and was using set of different classes with the only method findSingleton (). I have listed the differences between both classes:

1. In tutorial 2 we were using two classes Resource & SingletonResourceHome which can initialize single instance but in tutorial 3 we are importing different ser of classes which can creates multiple resources and assign unique key for later use.
   ```
   import org.globus.wsrf.ResourceKey;
   import org.globus.wsrf.impl.ResourceHomeImpl;
   import org.globus.wsrf.impl.SimpleResourceKey;
   ```
2. In tutorial 2 HelloWorldResourceHome was extending SingletonResourceHome, but in tutorial 3 we are extending ResourceHomeImpl
3. For tutorial 2 HelloWorldResourceHome had findSingleton () method which was returning Resource but in tutorial 3 we have create() with entirely different implementation which returns ResourceKey.

Below is the complete code from "HelloWorldResourceHome.java":

```
package uk.ac.dl.ws.service;
```

```
import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.impl.ResourceHomeImpl;
import org.globus.wsrf.impl.SimpleResourceKey;

public class HelloWorldResourceHome extends ResourceHomeImpl{

  public ResourceKey create() {
    try {
      HelloWorldResource hello = (HelloWorldResource) createNewInstance();
      hello.initialize();
      ResourceKey key = new SimpleResourceKey(keyTypeName, hello.getID());
      this.add(key, hello);
      return key;
    }
    catch (Exception e) {
      System.out.println("Exception when creating HelloWorld: ");
      e.printStackTrace();
      return null;
    }
  }
}
```

Third class to be modified is our HelloWorld.java, which is the class with business logic of our Web Service, this class has similar contents as of HelloWorld.java (from tutorial 2), the only difference is additional business method create() returning EndpointReferenceType which is declared in the WSDL file.

```
public EndpointReferenceType create(Create c) throws RemoteException {
  ResourceContext ctx = ResourceContext.getResourceContext();
  HelloWorldResourceHome home = (HelloWorldResourceHome) ctx.getResourceHome();

  // create a new HelloWorld Resource
  ResourceKey key = home.create();
  // form an EPR that points to the sticky note
  EndpointReferenceType epr;
  try {
    epr = AddressingUtils.createEndpointReference(ctx, key);
  }
  catch (Exception e) {
    throw new RemoteException("Could not form an EPR to new counter: " + e);
  }
  return epr;
}
```

Below is complete code from HelloWorld.java which needs no explaination and be sure to import appropriate packages related to our new target namespace in the WSDL file and EndpointReferenceType i.e.

```
import uk.ac.dl.wsrf.tutorial3.helloworld.*;
```

```
package uk.ac.dl.ws.service;

import java.rmi.RemoteException;
import org.apache.axis.message.addressing.EndpointReferenceType;
```

```
import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.utils.AddressingUtils;
import uk.ac.dl.wsrf.tutorial3.helloworld.*;

public class HelloWorld {

  public HelloWorld() throws RemoteException {

  }

  public String echo(String name) throws RemoteException {
    HelloWorldResource helloWorldResource = getResource();
    helloWorldResource.setNameRP(name);
    return "Hello " + name + " !";
  }

  public String getNameRP(GetNameRPRequest params) throws RemoteException {
    HelloWorldResource helloWorldResource = getResource();
    return helloWorldResource.getName();
    //return "This method is working Tutorial 3";
  }

  public HelloWorldResource getResource() throws RemoteException {
    Object resource = null;
    try {
      resource = ResourceContext.getResourceContext().getResource();
    } catch (Exception e) {
      throw new RemoteException("", e);
    }
    HelloWorldResource helloWorldResource = (HelloWorldResource) resource;
    return helloWorldResource;
  }

  public EndpointReferenceType create(Create c) throws RemoteException {
    ResourceContext ctx = ResourceContext.getResourceContext();
    HelloWorldResourceHome home = (HelloWorldResourceHome) ctx.getResourceHome();

    // create a new service
    ResourceKey key = home.create();
    // form an EPR that points to the service
    EndpointReferenceType epr;
    try {
      epr = AddressingUtils.createEndpointReference(ctx, key);
    }
    catch (Exception e) {
      throw new RemoteException("Could not form an EPR to new counter: " + e);
    }
    return epr;
  }
}
```


WSRF based HelloWorld with multiple instances

In HelloWorld.java; create() method first get hold of HelloWorldResourceHome through ResourceContext and calls its create method,

```
ResourceContext ctx = ResourceContext.getResourceContext();
HelloWorldResourceHome home = (HelloWorldResourceHome) ctx.getResourceHome();
ResourceKey key = home.create();
```

HelloWorldResourceHome.create() returns ResourceKey, which is used to create EndpointReferenceType using AddressingUtils.createEndpointReferenc(), which takes ResourceContext & ResourceKey as parameters.

Be sure about the last import statement, these are classes in the new package which matches the target namespace mentioned in our WSDL file.

targetNamespace="http://tutorial3.wsrf.dl.ac.uk/helloworld"

We can change this mapping from this "targetNamespace" to any required package by providing a simple text file called NStoPkg.properties, which, I have avoided in this simple tutorial.

Package uk.ac.dl.wsrf.tutorial3.helloworld.* has few classes which are used by client and service both, If you remember when we were changing the WSDL file, I talked about wrapper to represent void and empty parameter to method call. Our both methods *echo()* and *getNameRP()* return *String* but getNameRP() and create() doesn't take any parameter and this is wrapped in the empty dataType "*getNameRPRequest*" and "create". Stub will be created representing this data type and the name of stub generated is GetNameRPRequest.java and Create.java. It is important to understand as you have to import the related classes which are still not available and you should know the location and name of these classes by analysing the WSDL file at the time when Web Service is implemented.

Below is the table which describes all differences and similarities of Tutorial 3 and Tutorial 2 implementation.

Tutorial 3	Tutorial 2
package uk.ac.dl.ws.service; *import org.globus.wsrf.Resource;* *import org.globus.wsrf.ResourceProperties;* *import org.globus.wsrf.ResourceProperty;* *import org.globus.wsrf.ResourcePropertySet;* *import org.globus.wsrf.ResourceIdentifier;* *import org.globus.wsrf.impl.SimpleResourceProperty;* *import org.globus.wsrf.impl.SimpleResourcePropertySet;* *public class HelloWorldResource implements Resource, ResourceProperties, ResourceIdentifier {* */** the identifier of this service */* *private Object id;* */* Resource Property set */* *private ResourcePropertySet propSet;* */* Resource properties */* *private String name;* */** Resource property "name". */ // making it global* *private ResourceProperty nameRP;*	*package uk.ac.dl.ws.service;* *import org.globus.wsrf.Resource;* *import org.globus.wsrf.ResourceProperties;* *import org.globus.wsrf.ResourceProperty;* *import org.globus.wsrf.ResourcePropertySet;* *~~import org.globus.wsrf.impl.ReflectionResourceProperty;~~* *import org.globus.wsrf.impl.SimpleResourcePropertySet;* *public class HelloWorldResource implements Resource, ResourceProperties {* */* Resource Property set */* *private ResourcePropertySet propSet;* */* Resource properties */* *private String name;* *public void initialize() throws Exception {* */* Create RP set */* *this.propSet = new SimpleResourcePropertySet(* *HelloQNames.RESOURCE_PROPERTIES);* *~~/* Initialize the RP's */~~* *~~try {~~*

```
public void initialize() throws Exception {
  // choose an ID
  this.id = new Integer(hashCode());

  /* Create RP set */
  this.propSet = new SimpleResourcePropertySet(
    HelloQNames.RESOURCE_PROPERTIES);
  // create resource properties
  this.nameRP = new
SimpleResourceProperty(HelloQNames.RP_NAME);

   this.propSet.add(nameRP);
   setName("Asif Akram Tutorial 3");
   this.nameRP.add(name);
 }
 public String getName() {
  return name;
 }
 public void setName(String name) {
  this.name = name;
 }
 /* Required by interface ResourceProperties */
 public ResourcePropertySet getResourcePropertySet() {
  return this.propSet;
 }
 public void setNameRP(String nameValue) {
  setName(nameValue);
  this.nameRP.set(0, name);
 }
 public Object getID() {
  return id;
 }
}
```

```
    ResourceProperty nameRP = new
ReflectionResourceProperty(
        HelloQNames.RP_NAME, "name", this);
    this.propSet.add(nameRP);
    setName("Asif Akram Tutorial 2");

  }
  catch (Exception e) {
   throw new RuntimeException(e.getMessage());
  }
 }

 public String getName() {
  return name;
 }

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

 /* Required by interface ResourceProperties */
 public ResourcePropertySet getResourcePropertySet() {
  return this.propSet;
 }

}
```

```
package uk.ac.dl.ws.service;

import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.impl.ResourceHomeImpl;
import org.globus.wsrf.impl.SimpleResourceKey;

public class HelloWorldResourceHome extends ResourceHomeImpl{

 public ResourceKey create() {
  try {
   HelloWorldResource hello = (HelloWorldResource)
       createNewInstance();
   hello.initialize();
   ResourceKey key = new
       SimpleResourceKey(keyTypeName, hello.getID());
   this.add(key, hello);
   return key;
  }
  catch (Exception e) {
   System.out.println("Exception when creating HelloWorld: ");
   e.printStackTrace();
   return null;
  }
 }
}
```

```
package uk.ac.dl.ws.service;

import org.globus.wsrf.Resource;
import org.globus.wsrf.impl.SingletonResourceHome;

public class HelloWorldResourceHome extends
SingletonResourceHome{

 public Resource findSingleton (){
  try {
   HelloWorldResource helloWorldResource = new
       HelloWorldResource();
   helloWorldResource.initialize();
   return helloWorldResource;
  } catch (Exception e){
   e.printStackTrace();
   return null;
  }
 }
}
```

```
package uk.ac.dl.ws.service;

import java.rmi.RemoteException;
```

```
package uk.ac.dl.ws.service;

import java.rmi.RemoteException;
```


WSRF based HelloWorld with multiple instances

```
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.utils.AddressingUtils;
import uk.ac.dl.wsrf.tutorial3.helloworld.*;

public class HelloWorld {
  public HelloWorld() throws RemoteException { }

  public String echo(String name) throws RemoteException {
    HelloWorldResource helloWorldResource = getResource();
    helloWorldResource.setNameRP(name);
    return "Hello " + name + " !";
  }
  public String getNameRP(GetNameRPRequest params) throws
RemoteException {
    HelloWorldResource helloWorldResource = getResource();
    return helloWorldResource.getName();
  }
  public HelloWorldResource getResource() throws
RemoteException {
    Object resource = null;
    try {
      resource = ResourceContext.getResourceContext().getResource();
    } catch (Exception e) {
      throw new RemoteException("", e);
    }
    HelloWorldResource helloWorldResource =
        (HelloWorldResource) resource;
    return helloWorldResource;
  }

  public EndpointReferenceType create(Create c) throws
RemoteException {
    ResourceContext ctx = ResourceContext.getResourceContext();
    HelloWorldResourceHome home =
        (HelloWorldResourceHome) ctx.getResourceHome();

    // create a new service
    ResourceKey key = home.create();
    // form an EPR that points to the service
    EndpointReferenceType epr;
    try {
      epr = AddressingUtils.createEndpointReference(ctx, key);
    }
    catch (Exception e) {
      throw new RemoteException("Could not form an EPR " + e);
    }
    return epr;
  }
}
```

```
import org.globus.wsrf.ResourceContext;
import uk.ac.dl.wsrf.tutorial2.helloworld.*;

public class HelloWorld {

  public HelloWorld() throws RemoteException {
  }

  public String echo (String name) throws RemoteException {
    HelloWorldResource helloWorldResource = getResource();
    helloWorldResource.setName(name);
    return "Hello " + name + " !";
  }

  public String getNameRP(GetNameRPRequest params) throws
     RemoteException {
    HelloWorldResource helloWorldResource = getResource();
    return helloWorldResource.getName();
  }

  public HelloWorldResource getResource() throws
RemoteException {
    Object resource = null;
    try {
      resource =
RemoteContext.getResourceContext().getResource();
    } catch (Exception e) {
      throw new RemoteException("", e);
    }
    HelloWorldResource helloWorldResource =
        (HelloWorldResource) resource;
    return helloWorldResource;
  }
}
```


Modified Deployment Descriptor:

Have we finished modification to our WSRF Web Service? Not yet … Don't forget we need Axis Engine specific dependent Deployment Descriptor to deploy our Web Service. This WSDD is very similar to previous WSDD only change is the change in the service name.

```
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:aggr="http://mds.globus.org/aggregator/types"
```

```
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <service name="HelloWorldService3" provider="Handler" use="literal" style="document">
    <parameter name="providers" value="GetRPProvider"/>
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="activateOnStartup" value="true"/>
    <parameter name="className" value=" uk.ac.dl.ws.service.HelloWorld "/>
    <wsdlFile>share/schema/tutorial/HelloWorld3_service.wsdl</wsdlFile>
  </service>
</deployment>
```

Other than just changing the name of our Service we have to mention the location of Final WSDL file which will be created by Globus when we will run Ant script. The final WSDL file will have similar name as our incomplete WSDL with suffix "_service.wsdl". In our WSDL file is helloworld2_port_type.wsdl therefore final WSDL file is HelloWorld2_service.wsdl. This is very important change; otherwise it will refer to WSDL file from previous tutorial and will give errors. In fact this is the name of WSDL file we have mentioned in the Ant script. If you remember we have changed the default script in last tutorial and this time the change will be:

```
<property name="binding.root" value="HelloWorld3"/>
```

Modified Deployment "jndi-config":

The last thing we need is "deploy-jndi-config.xml". This file is related to Tomcat Server and tells the server about the service and the class which provides the implementation of service. This file is very simple just like previous tutorial.

```
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
  <service name="HelloWorldService3">

    <resource
      name="home" type="uk.ac.dl.ws.service.HelloWorldResourceHome">
      <resourceParams>
        <parameter>
          <name>factory</name>
          <value>org.globus.wsrf.jndi.BeanFactory</value>
        </parameter>
        <parameter>
          <name>resourceClass</name>
          <value>uk.ac.dl.ws.service.HelloWorldResource</value>
        </parameter>
        <parameter>
          <name>resourceKeyName</name>
          <value>{http://tutorial3.wsrf.dl.ac.uk/helloworld}MyNameKey</value>
        </parameter>
        <parameter>
          <name>resourceKeyType</name>
          <value>java.lang.Integer</value>
        </parameter>
      </resourceParams>
```

```
    </resource>
  </service>
</jndiConfig>
```

The important bits about this "deploy-jndi-config.xml" are adding three new parameters:

1. resourceClass class implementing resource.
2. resourceKeyName can be any qualified name.
3. resourceKeyType should be of same data type as used in resourceClass to initialize "id"
   this.id = new Integer(hashCode());

Modified Implementation of our Client:

Last thing to do is to write Client to test our WSRF Web Service. Below is the complete code of Client3.java followed by discussing important parts, notice that Client3.java is in package "uk.ac.dl.ws.service.*client*" and our service HelloWorld.java is in package "*package* uk.ac.dl.ws.service". This is not the requirement of WSRF or Globus implementation of WSRF, in fact it is restriction due to the Ant "build.xml" file which will be used later to generate stubs, compile service and deploy to the container. You can change "build.xml" to adjust changes made in your package structure. We have to put all java classes and additional files like WSDL and "deploy-jndi-config.xml" in specific directories as required by Ant "build.xml". Ant "build.xml" is so handy and useful that following few restrictions is still worthy, as it hides all dirty work of setting class path, copying files from different locations and above all making all imported files in our WSDL available to our WSDL file.

```
package uk.ac.dl.ws.service.client;

import org.oasis.wsrf.properties.WSResourcePropertiesServiceAddressingLocator;
import org.oasis.wsrf.properties.GetResourceProperty;
import org.oasis.wsrf.properties.GetResourcePropertyResponse;
import org.apache.axis.message.addressing.EndpointReferenceType;

import org.apache.commons.cli.ParseException;
import org.apache.commons.cli.CommandLine;

import org.globus.wsrf.client.BaseClient;
import org.globus.wsrf.encoding.ObjectSerializer;
import org.globus.wsrf.encoding.SerializationException;
import org.globus.wsrf.utils.AnyHelper;
import org.globus.wsrf.utils.FaultHelper;

import java.io.FileWriter;
import java.io.IOException;

import java.util.List;
import java.util.Random;

import javax.xml.rpc.Stub;
import uk.ac.dl.ws.service.HelloQNames;
import uk.ac.dl.wsrf.tutorial3.helloworld.service.*;
import uk.ac.dl.wsrf.tutorial3.helloworld.*;

public class Client3 extends BaseClient {
  final static HelloWorld3ServiceAddressingLocator locator =  new HelloWorld3ServiceAddressingLocator();

  public static void main(String[] args) {
    Client3 client = new Client3();

    // first, parse the commandline
    try {
```

```
      CommandLine line = client.parse(args);
    }
    catch (ParseException e) {
      System.err.println("Parsing failed: " + e.getMessage());
      System.exit(1);
    }
    catch (Exception e) {
      System.err.println("Error: " + e.getMessage());
      System.exit(1);
    }

    try {
      HelloWorldPortType port = locator.getHelloWorldPortTypePort(client.getEPR());
      client.setOptions( (Stub) port);

      EndpointReferenceType epr = port.create(new Create());
      System.out.println("new resource created...");

      // To Test SOAP messages in TCP Monitor
      System.out.println(epr.getAddress());
      epr.getAddress().setPort(8082);
      System.out.println(epr.getAddress());

      // Setting EndPoint for further use
      port = locator.getHelloWorldPortTypePort(epr);

      GetResourcePropertyResponse response =  port.getResourceProperty(HelloQNames.RP_NAME);
      System.out.println(AnyHelper.toSingleString(response));

      String result = port.echo("Rob Allan First Time Tutorial-3");
      response = port.getResourceProperty(HelloQNames.RP_NAME);
      System.out.println(AnyHelper.toSingleString(response));

      result = port.echo("Rob Allan Second Time Tutorial-3");
      response =  port.getResourceProperty(HelloQNames.RP_NAME);
      System.out.println(AnyHelper.toSingleString(response));

// alternative way of retrieving the value of "name" variable through business method
      System.out.println(port.getNameRP(new GetNameRPRequest()));    }
    catch (Exception e) {
      if (client.isDebugMode()) {
        FaultHelper.printStackTrace(e);
      }
      else {
        System.err.println("Error: " + FaultHelper.getMessage(e));
      }
    }
  }
}
```

Implementation of Client once again starts with importing couple of Globus dependent java classes along with importing few classes which are not yet available and will created by Ant build file. One of them is to locate our Web Service *"HelloWorldServiceAddressingLocator"* which is in the package *"uk.ac.dl.wsrf.tutorial3.helloworld.service"* and the *"HelloWorldPortType"* which is the interface with all three methods available in our main Web Service. *"HelloWorldPortType" is available in the package "import uk.ac.dl.wsrf.tutorial3.helloworld".* Our client implementation is using HelloWorldServiceAddressingLocator to create custom stub HelloWorldPortType, through this stub we have access to all operations exposed by WSDL as shown in following code fragment;

```
HelloWorldServiceAddressingLocator mylocator =  new HelloWorldServiceAddressingLocator();
```

```
HelloWorldPortType myPort = mylocator.getHelloWorldPortTypePort (client.getEPR());
String result = myPort.echo("Rob Allan First Time Tutorial-3");
```

If you compare the implementation of Client3.java with the implementation of Client1.java in tutorial 2, you will notice that we are importing too many packages, which were not used in the Client1.java. We in fact don't need to import all those packages if we are just creating Singleton resource.

Modified "post-deploy":

Now everything is available and we have to build the WSRF Web Service. For build purposes we will use build tool "ANT" and the build.xml provided by the Globus. To make life easy to test web service we need last file "post-deploy.xml" which will create script to run the client, without setting class path, without worrying about the generated stubs and location of compiled stubs. This xml file is generated when everything goes well, below are the contents of file "post-deploy.xml".

```
<project default="all" basedir=".">
   <property environment="env"/>
   <property file="build.properties"/>
   <property file="${user.home}/build.properties"/>
   <property name="env.GLOBUS_LOCATION" value="."/>
   <property name="deploy.dir" location="${env.GLOBUS_LOCATION}"/>
   <property name="abs.deploy.dir" location="${deploy.dir}"/>
   <property name="build.launcher"
     location="${abs.deploy.dir}/share/globus_wsrf_common/build-launcher.xml"/>

   <target name="setup">
    <ant antfile="${build.launcher}"
       target="generateLauncher">
      <property name="launcher-name" value="helloworld3"/>
      <property name="class.name"  value="uk.ac.dl.ws.service.client.Client3"/>
     </ant>
  </target>
</project>
```

This file is quite simple, and most of time remains same for most of clients. There are two adjustments to be made for each client and location of those adjustments is shown in bold. Name of the generated script: ***<property name="launcher-name" value="helloworld3"/>*** and location and name of java client of WSRF Web Service. In our case client is "*Client3.java*" in package "***uk.ac.dl.ws.service.client":***

```
<property name="class.name"    value="uk.ac.dl.ws.service.client.Client3"/>
```

Tutorial 3: *"post-deploy.xml"*	Tutorial 2: *"post-deploy.xml"*
<project default="all" basedir="."> <property environment="env"/> <property file="build.properties"/> <property file="${user.home}/build.properties"/> <property name="env.GLOBUS_LOCATION" value="."/> <property name="deploy.dir" location="${env.GLOBUS_LOCATION}"/> <property name="abs.deploy.dir" location="${deploy.dir}"/> <property name="build.launcher" location="${abs.deploy.dir}/share/globus_wsrf_common/build-launcher.xml"/>	<project default="all" basedir="."> <property environment="env"/> <property file="build.properties"/> <property file="${user.home}/build.properties"/> <property name="env.GLOBUS_LOCATION" value="."/> <property name="deploy.dir" location="${env.GLOBUS_LOCATION}"/> <property name="abs.deploy.dir" location="${deploy.dir}"/> <property name="build.launcher" location="${abs.deploy.dir}/share/globus_wsrf_common/build-launcher.xml"/>

```
  <target name="setup">
    <ant antfile="${build.launcher}"
      target="generateLauncher">
      <property name="launcher-name" value="helloworld3"/>
      <property name="class.name"
        value="uk.ac.dl.ws.service.client.Client3"/>
    </ant>
  </target>
</project>
```

```
  <target name="setup">
    <ant antfile="${build.launcher}"
      target="generateLauncher">
      <property name="launcher-name" value="helloworld2"/>
      <property name="class.name"
        value="uk.ac.dl.ws.service.client.Client1"/>
    </ant>
  </target>
</project>
```

Modified Ant Script:

Now we have to use Ant build file, but that Ant file also needs few changes related to our Web Service. These changes are very small and those changes are the last step required to finish the whole implementation.

Once again I have copied the complete build file and highlighted the changes related to Tutorial 3 in Blue Font:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all" name="globus_tutorial3_ helloworld ">

  <property environment="env"/>

  <property file="build.properties"/>
  <property file="${user.home}/build.properties"/>
  <property location="${env.GLOBUS_LOCATION}" name="deploy.dir"/>
  <property name="base.name" value="tutorial3_helloworld"/>
  <property name="package.name" value="globus_${base.name}"/>
  <property name="jar.name" value="${package.name}.jar"/>
  <property name="gar.name" value="${package.name}.gar"/>
  <property location="build" name="build.dir"/>
  <property location="build/classes" name="build.dest"/>
  <property location="build/lib" name="build.lib.dir"/>
  <property location="build/stubs" name="stubs.dir"/>
  <property location="build/stubs/src" name="stubs.src"/>
  <property location="build/stubs/classes" name="stubs.dest"/>
  <property name="stubs.jar.name" value="${package.name}_stubs.jar"/>
  <property location="${deploy.dir}/share/globus_wsrf_common/build-packages.xml"
      name="build.packages"/>
  <property location="${deploy.dir}/share/globus_wsrf_tools/build-stubs.xml" name="build.stubs"/>
  <property name="java.debug" value="on"/>
  <property name="test-reports.dir" value="test-reports"/>
  <property name="junit.haltonfailure" value="true"/>

  <property location="${deploy.dir}/share/schema" name="schema.src"/>
  <property location="${build.dir}/schema" name="schema.dest"/>

  <property name="garjars.id" value="garjars"/>
  <fileset dir="${build.lib.dir}" id="garjars"/>

  <property name="garschema.id" value="garschema"/>
  <fileset dir="${schema.dest}" id="garschema" includes="tutorial/*"/>
```

```
<property name="garetc.id" value="garetc"/>
<fileset dir="etc" id="garetc"/>

<target name="init">
  <mkdir dir="${build.dir}"/>
  <mkdir dir="${build.dest}"/>
  <mkdir dir="${build.lib.dir}"/>

  <mkdir dir="${stubs.dir}"/>
  <mkdir dir="${stubs.src}"/>
  <mkdir dir="${stubs.dest}"/>

  <mkdir dir="${schema.dest}"/>

  <copy toDir="${schema.dest}">
    <fileset dir="${schema.src}" casesensitive="yes">
      <include name="wsrf/**/*"/>
      <include name="ws/**/*"/>
    </fileset>
  </copy>
  <copy toDir="${schema.dest}">
    <fileset dir="schema/" casesensitive="yes">
      <include name="tutorial/*"/>
    </fileset>
  </copy>

  <available file="${stubs.dest}/org.globus.wsrf.mds.aggregator" property="stubs.present" type="dir"/>
</target>

<target name="bindings" depends="init">
<ant antfile="${build.stubs}" target="generateBinding">
    <property name="source.binding.dir"
      value="${schema.dest}/tutorial"/>
    <property name="target.binding.dir"
      value="${schema.dest}/tutorial"/>
    <property name="binding.root" value="HelloWorld3"/>
    <property name="porttype.wsdl" value="helloworld3_port_type.wsdl"/>
  </ant>

</target>
<target name="stubs" unless="stubs.present" depends="bindings">
  <ant antfile="${build.stubs}" target="generateStubs">
    <property location="${schema.dest}/tutorial" name="source.stubs.dir"/>
    <property name="wsdl.file" value="HelloWorld3_service.wsdl"/>
    <property location="${stubs.src}" name="target.stubs.dir"/>
  </ant>
</target>

<target depends="stubs" name="compileStubs">
  <delete dir="${stubs.src}/org/apache"/>
  <javac debug="${java.debug}" destdir="${stubs.dest}" srcdir="${stubs.src}">
    <include name="**/*.java"/>
```

```
      <classpath>
        <fileset dir="${deploy.dir}/lib">
          <include name="*.jar"/>
        </fileset>
      </classpath>
    </javac>
  </target>

  <target depends="compileStubs" name="compile">
    <javac debug="${java.debug}" destdir="${build.dest}" srcdir="src">
      <include name="**/*.java"/>
      <classpath>
        <pathelement location="${stubs.dest}"/>
        <fileset dir="${deploy.dir}/lib">
          <include name="*.jar"/>
        </fileset>
      </classpath>
    </javac>
  </target>

  <target depends="compileStubs" name="jarStubs">
    <jar basedir="${stubs.dest}" destfile="${build.lib.dir}/${stubs.jar.name}"/>
  </target>

  <target depends="compile" name="jar">
    <jar basedir="${build.dest}" destfile="${build.lib.dir}/${jar.name}"/>
  </target>

  <target depends="jar, jarStubs" name="dist">
    <ant antfile="${build.packages}" target="makeGar">
      <property name="gar.name" value="${build.lib.dir}/${gar.name}"/>
      <reference refid="${garjars.id}"/>
      <reference refid="${garschema.id}"/>
      <reference refid="${garetc.id}"/>
    </ant>
  </target>

  <target name="clean">
    <delete dir="tmp"/>
    <delete dir="${build.dir}"/>
    <delete file="${gar.name}"/>
    <delete dir="${test-reports.dir}"/>
  </target>

  <target depends="dist" name="deploy">
    <ant antfile="${build.packages}" target="deployGar">
      <property name="gar.name" value="${build.lib.dir}/${gar.name}"/>
      <property name="gar.id" value="${package.name}"/>
      <!-- <property name="noSchema" value="true"/> -->
    </ant>
  </target>
```

```
<target name="undeploy">
  <ant antfile="${build.packages}" target="undeployGar">
    <property name="gar.id" value="${package.name}"/>
  </ant>
</target>

<target depends="deploy" name="all"/>

<target depends="compile" name="test">
  <mkdir dir="${test-reports.dir}"/>
  <junit fork="yes" haltonfailure="${junit.haltonfailure}" printsummary="yes" timeout="600000">
    <classpath>
      <pathelement location="${build.dest}"/>
      <pathelement location="${deploy.dir}"/>
      <pathelement location="${deploy.dir}/etc/wsrf-bundles.properties"/>
      <fileset dir="${deploy.dir}/lib">
        <include name="*.jar"/>
      </fileset>
    </classpath>
    <formatter type="xml"/>
    <batchtest todir="${test-reports.dir}">
      <fileset dir="${build.dest}">
        <include name="**/*Test.class"/>
      </fileset>
    </batchtest>
  </junit>
</target>
</project>
```

There are only 4 minor changes in the Ant Script from Tutorial 2; those changes are highlighted in the Blue Font. Changes in the script are more related to changing the name of WSDL file used for Tutorial 2 i.e. "*helloworld2_port_type.wsdl*", and the name which Ant Script will use to generate complete WSDL file,

```
<property name="binding.root" value="HelloWorld3"/>
<property name="porttype.wsdl" value="helloworld3_port_type.wsdl"/>
```

Before we can run Ant script we have to put them at proper location where Ant build can locate them. HelloWorld220605A is the main directory containing all files under one umbrella, our Web Service implementation *HelloWorld.java* will be in the directory structure according to package statement i.e. "*HelloWorld220605A\source\src\uk\ac\dl\ws\service*" our *Client1.java* will be the directory "*HelloWorld220605A\source\src\uk\ac\dl\ws\service\client*" (2). Our WSDL file "*helloworld2_port_type.wsdl*" is located at "*HelloWorld220605A\source\schema\tutorial*" and source directory contains "deploy-server.wsdd", "deploy-jndi-config.xml" and "build.xml". File "post-deploy.xml", which; is used for creating client script is in directory "HelloWorld220605A*source\etc*".

Compiling and Deploying Modified WSRF Web Service:

I am using Windows Operating System and have installed only WS-Core component of Globus Toolkit. You have to set an environment variable called GLOBUS_LOCATION referencing the installation of Globus Toolkit. In my case it is:

set GLOBUS_LOCATION= *E:\GlobusWeek\gt-install*

Open two Command Prompt one for starting Globus Tomcat Server and other for deploying Web Service and testing client. In one go to directory E:\GlobusWeek\HelloWorld220605A\source and run build file to create stubs, compile all java files, and deploy Web Service.

ant clean
ant deploy

If everything goes fine then after 10 seconds you will see the final outcome similar to:

```
generateWindows:
  [echo] Creating Windows launcher script helloworld3
  [copy] Copying 1 file to E:\GlobusWeek\gt-install\bin
 [delete] Deleting: E:\GlobusWeek\HelloWorld220605A\source\tmp\gar\etc\post-d
[delete] Deleting directory E:\GlobusWeek\HelloWorld220605A\source\tmp\gar
Total time: 8 seconds
```

In second Window go to directory "E:\GlobusWeek\gt-install" and run command,

bin\globus-start-container –nosec

which will start the container.

Testing Modified WSRF Web Service:

If you remember in the file "post-deploy.xml" we have mentioned helloworld3 as name of script to be generated to run the client, which is confirmed by the final outcome of the Ant build file. Now it is time to run the client:

```
%GLOBUS_LOCATION%\bin\helloworld3 -s http://localhost:8082/wsrf/services/HelloWorldService3
new resource created...
<ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">Asif Akram from Tutorial 3</ns1:name>
<ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">Rob Allan First Time Tutorial-3</ns1:name>
<ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">Rob Allan Second Time Tutorial-3</ns1:name>
```

"*Asif Akram from Tutorial 3*" is the initial value assigned to our variable "name" and "*Rob Allan First Time Tutorial-3*" is the value assigned to variable "name" from "Client3.java". Client3.java calls "echo" method twice and second time it passes the value "*Rob Allan Second Time Tutorial-3*". If you will run the script again then the out come will be:

```
%GLOBUS_LOCATION%\bin\helloworld3 -s http://localhost:8082/wsrf/services/HelloWorldService3
new resource created...
<ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">Asif Akram from Tutorial 3</ns1:name>
<ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">Rob Allan First Time Tutorial-3</ns1:name>
<ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">Rob Allan Second Time Tutorial-3</ns1:name>
```

No matter how many times we run client, every time new instance will be created, that's why every time our variable "name" has new value.

SOAP Request	Soap Response
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"> <soapenv:Header> <wsa:MessageID soapenv:mustUnderstand="0"> uuid:06d3b700-eee5-11d9-9df0-bd51fff218a3 </wsa:MessageID> <wsa:To soapenv:mustUnderstand="0"> http://localhost:8082/wsrf/services/HelloWorldService3 </wsa:To> <wsa:Action soapenv:mustUnderstand="0"> http://tutorial3.wsrf.dl.ac.uk/helloworld/HelloWorldPortType/CreateRequest </wsa:Action> <wsa:From soapenv:mustUnderstand="0"> <wsa:Address>	<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"> <soapenv:Header> <wsa:MessageID soapenv:mustUnderstand="0"> uuid:06e1e7d0-eee5-11d9-ad05-8639052688f4 </wsa:MessageID> <wsa:To soapenv:mustUnderstand="0"> http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous </wsa:To> <wsa:Action soapenv:mustUnderstand="0"> http://tutorial3.wsrf.dl.ac.uk/helloworld/HelloWorldPortType/CreateResponse </wsa:Action> <wsa:From soapenv:mustUnderstand="0"> <wsa:Address> http://localhost:8082/wsrf/services/HelloWorldService3

<table>
<tr>
<td>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
</wsa:Address>
</wsa:From>
</soapenv:Header>
<soapenv:Body>
<Create xmlns="http://tutorial3.wsrf.dl.ac.uk/helloworld"/>
</soapenv:Body>
</soapenv:Envelope></td>
<td></wsa:Address>
</wsa:From>
<wsa:RelatesTo RelationshipType="wsa:Reply"
soapenv:mustUnderstand="0">
uuid:06d3b700-eee5-11d9-9df0-bd51fff218a3
</wsa:RelatesTo>
</soapenv:Header>
<soapenv:Body>
<CreateResponse
xmlns="http://tutorial3.wsrf.dl.ac.uk/helloworld">
<wsa:Address>
http://193.62.113.83:8080/wsrf/services/HelloWorldService3
</wsa:Address>
<wsa:ReferenceProperties>
<MyNameKey>9296787</MyNameKey>
</wsa:ReferenceProperties>
<wsa:ReferenceParameters/>
</CreateResponse>
</soapenv:Body>
</soapenv:Envelope></td>
</tr>
<tr>
<td><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
<soapenv:Header>
<wsa:MessageID soapenv:mustUnderstand="0">
uuid:07080d70-eee5-11d9-9df0-bd51fff218a3
</wsa:MessageID>
<wsa:To soapenv:mustUnderstand="0">
http://193.62.113.83:8082/wsrf/services/HelloWorldService3
</wsa:To>
<wsa:Action soapenv:mustUnderstand="0">
http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourceProperty
</wsa:Action>
<wsa:From soapenv:mustUnderstand="0">
<wsa:Address>
http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
</wsa:Address>
</wsa:From>
<ns1:MyNameKey
xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld"
soapenv:mustUnderstand="0">
9296787
</ns1:MyNameKey>
</soapenv:Header>
<soapenv:Body>
<GetResourceProperty
xmlns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd"
xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">
ns1:name
</GetResourceProperty>
</soapenv:Body>
</soapenv:Envelope></td>
<td><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
<soapenv:Header>
<wsa:MessageID soapenv:mustUnderstand="0">
uuid:070cc860-eee5-11d9-ad05-8639052688f4
</wsa:MessageID>
<wsa:To soapenv:mustUnderstand="0">
http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
</wsa:To>
<wsa:Action soapenv:mustUnderstand="0">
http://docs.oasis-open.org/wsrf/2004/06/.....
</wsa:Action>
<wsa:From soapenv:mustUnderstand="0"
xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">
<wsa:Address>
http://193.62.113.83:8082/wsrf/services/HelloWorldService3
</wsa:Address>
<wsa:ReferenceProperties>
<ns1:MyNameKey soapenv:mustUnderstand="0"
xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
9296787
</ns1:MyNameKey>
</wsa:ReferenceProperties>
</wsa:From>
<wsa:RelatesTo RelationshipType="wsa:Reply"
soapenv:mustUnderstand="0">
uuid:07080d70-eee5-11d9-9df0-bd51fff218a3
</wsa:RelatesTo>
</soapenv:Header>
<soapenv:Body>
<GetResourcePropertyResponse
xmlns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd">
<ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">
Asif Akram from Tutorial 3
</ns1:name>
</GetResourcePropertyResponse>
</soapenv:Body>
</soapenv:Envelope></td>
</tr>
</table>

Let see SOAP messages to understand the working of WSRF. I have changed the port number of the service in the Client3.java to use TCP Monitor:

```
// To Test SOAP messages in TCP Monitor
```

WSRF based HelloWorld with multiple instances

```
System.out.println(epr.getAddress());
epr.getAddress().setPort(8082);
System.out.println(epr.getAddress());

// Setting EndPoint for further use
port = locator.getHelloWorldPortTypePort(epr);
```

First SOAP request is very simple which is calling business method create() with target namespace, but in response to this call WSRF service is returning EndpointReferenceType.

```
<wsa:Address>
    http://193.62.113.83:8080/wsrf/services/HelloWorldService3
</wsa:Address>
<wsa:ReferenceProperties>
  <MyNameKey>9296787</MyNameKey>
</wsa:ReferenceProperties>
<wsa:ReferenceParameters/>
```

If you remember from previous tutorials EndpointReferenceType is wrapper around WSRF Service URL and unique ID. The name of unique ID is MyNameKey, which was mentioned in the deploy-jndi.xml as

```
<parameter>
  <name>resourceKeyName</name>
  <value>{http://tutorial3.wsrf.dl.ac.uk/helloworld}MyNameKey</value>
</parameter>
```

For further interaction with WSRF service client has to send EndpointReferenceType to interact with the same instance as it is done in the second SOAP request, EndpointReferenceType information is sent in the SOAP header.

```
<ns1:MyNameKey xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld" soapenv:mustUnderstand="0">
    9296787
</ns1:MyNameKey>
```

In response to second SOAP request, WSRF Service is sending EndpointReferenceType in the SOAP Response header and result of the called business method is in the body of SOAP Response.

```
<soapenv:Header>
    …..
    …..
    <wsa:From soapenv:mustUnderstand="0"  xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">
      <wsa:Address>
        http://193.62.113.83:8082/wsrf/services/HelloWorldService3
     </wsa:Address>
      <wsa:ReferenceProperties>
        <ns1:MyNameKey soapenv:mustUnderstand="0"  xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld"
          xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
            9296787
        </ns1:MyNameKey>
      </wsa:ReferenceProperties>
    </wsa:From>
    …..
    …..
  </soapenv:Header>
  <soapenv:Body>
    <GetResourcePropertyResponse
```

```
        xmlns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
       <ns1:name xmlns:ns1="http://tutorial3.wsrf.dl.ac.uk/helloworld">
            Asif Akram from Tutorial 3
       </ns1:name>
     </GetResourcePropertyResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Each client will have its own unique ID, and if two clients have same unique ID then it means that they are sharing the same instance.

Note: QName in HelloWorld.java and Client1.java should be same, otherwise it will be error, surprisingly if QName in HelloWorld.java and WSDL are different, client still works. My understanding is all three should be exactly same, better to test again before conclusion. Technically al three should be same but Globus implementation of WSRF doesn't impost any constraint on the server side implementation to use the names as mentioned in the WSDL file. Client can only guess the QName of Resource Properties from the WSDL file therefore keeps them similar otherwise no client can interact with your service.