Asif Akram
a.akram@dl.ac.uk

In this tutorial one Instance Service will manage multiple resources; there is no limit how many resources single Instance Service can manage. Resource sharing among multiple Instance Services and single Instance Service managing multiple resources gives new dimension to the effectiveness of WSRF; which leads to limitless different scenarios. This tutorial is similar to last two tutorials and we will be using the same example of PersonResource with slight changes. The main difference is splitting Person Resource into two separate resources NameResource and AddressResource and modified WSDL for Instance Service. This tutorial is using stuff from Tutorial-9 and Tutorial-10 (use of InitialContext) so it is recommended to first read/understand those two tutorials.

WSDL for Factory Service and Instance Service

We will start discussion from the WSDL which is the best practice at least what I believe; before working on WSRF I always felt more comfortable by implementing the Service and generating the WSDL from the service implementation but that approach was headache; the main reason was the way Axis generates WSDL is not fully WS-I Basic Profile compatible and interoperability with other Web Services Framework is big issue which is against the spirit of Web Services. In WS-Core and Apollo (Apache implementation of WSRF) starting with the WSDL is only choice and there is no other possibility.

WSDL for Factory Service

WSDL for our Factory Service is similar to the WSDL from the Tutorial-9 with single createResource method but this time it returns the array of EndPointReference of the Resource and Instance Service. Our Instance Service is managing multiple resources thus depending on which resource is instantiated it should return the corresponding EndPointReference, but this requires smart createResource operation which should parse the request message and instantiate the appropriate resource and return the corresponding EndPointReference. I have kept the things simple and createResource takes no parameter and instantiate both resources i.e. NameResource and AddressResource and thus returning two EndPointReference which means array of EndPointReference with the length 2 (i.e. index number 0 and 1). Below is the complete WSDL for Factory Service which is saved as "*FactoryPersonMultiple.wsdl*":

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="FactoryPersonMultipleService"
    targetNamespace="http://sharing.wsrf.dl.ac.uk/multiple/person"
    xmlns:tns="http://sharing.wsrf.dl.ac.uk/multiple/person"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
<xsd:schema targetNamespace="http://sharing.wsrf.dl.ac.uk/multiple/person"
    xmlns:tns="http://sharing.wsrf.dl.ac.uk/multiple/person"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        schemaLocation="../ws/addressing/WS-Addressing.xsd"/>

      <xsd:element name="createResource">
            <xsd:complexType/>
      </xsd:element>
```

```
      <xsd:element name="createResourceResponse">
          <xsd:complexType>
              <xsd:sequence>
                <xsd:element ref="wsa:EndpointReference" minOccurs="0" maxOccurs="unbounded"/>
              </xsd:sequence>
          </xsd:complexType>
      </xsd:element>

</xsd:schema>
</types>

<message name="CreateResourceRequest">
      <part name="request" element="tns:createResource"/>
</message>
<message name="CreateResourceResponse">
      <part name="response" element="tns:createResourceResponse"/>
</message>

<portType name="PersonMultipleFactoryPortType">
      <operation name="createResource">
            <input message="tns:CreateResourceRequest"/>
            <output message="tns:CreateResourceResponse"/>
      </operation>
</portType>

</definitions>
```

WSDL for Instance Service

Our single Instance Service is managing multiple Resources i.e. *"NameResource"* and *"AddressResource"* thus exposing all four business methods changeName, getNameRP, changeAddress and getAddressRP which were in two different WSDL's in last two tutorials. This WSDL is basically the union of two WSDL from last two tutorials with only difference that it has split the single Person Resource into two resources. Below is the complete WSDL for our Instance Service which is saved as "*PersonMultiple.wsdl*":

```
<definitions name="PersonMultipleService"
    targetNamespace="http://sharing.wsrf.dl.ac.uk/multiple/person"
    xmlns:tns="http://sharing.wsrf.dl.ac.uk/multiple/person"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    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:myTypes="http://test.wsrf.resource.sharing"
    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-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"/>
```

```
<types>
  <schema
      targetNamespace="http://sharing.wsrf.dl.ac.uk/multiple/person"
      xmlns:tns="http://sharing.wsrf.dl.ac.uk/multiple/person"
      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://test.wsrf.resource.sharing"
      schemaLocation="types.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="PersonNameResourcePropertiesSet">
      <complexType>
        <sequence>
          <element ref="myTypes:myName"/>
        </sequence>
      </complexType>
    </element>

    <element name="PersonAddressResourcePropertiesSet">
      <complexType>
        <sequence>
          <element ref="myTypes:myAddress"/>
        </sequence>
      </complexType>
    </element>

    <!-- Elements related to changeName Method -->
    <element name="changeName" ref="myTypes:myName" />
    <element name="changeNameResponse" type="xsd:string" />

    <!-- Elements related to getNameRP Method -->
    <element name="getNameRP" >
      <complexType/>
     </element>
     <element name="getNameRPResponse" ref="myTypes:myName"/>

    <!-- Elements related to changeAddress Method -->
    <element name="changeAddress" ref="myTypes:myAddress" />
    <element name="changeAddressResponse" type="xsd:string" />

    <!-- Elements related to getAddressRP Method -->
    <element name="getAddressRP" >
      <complexType/>
     </element>
     <element name="getAddressRPResponse" ref="myTypes:myAddress"/>
  </schema>
</types>

<!-- Messages related to changeName Method -->
<message name="ChangeNameRequest">
  <part name="ChangeNameRequest"  element="tns:changeName" />
</message>
<message name="ChangeNameResponse">
  <part name="ChangeNameResponse" element="tns:changeNameResponse" />
</message>
```

```
<!-- Messages related to getNameRP Method -->
<message name="GetNameRPRequest">
  <part name="GetNameRPRequest" element="tns:getNameRP" />
</message>
<message name="GetNameRPResponse">
  <part name="GetNameRPResponse" element="tns:getNameRPResponse" />
</message>

<!-- Messages related to changeAddress Method -->
<message name="ChangeAddressRequest">
  <part name="ChangeAddressRequest"  element="tns:changeAddress" />
</message>
<message name="ChangeAddressResponse">
  <part name="ChangeAddressResponse" element="tns:changeAddressResponse" />
</message>

<!-- Messages related to getAddressRP Method -->
<message name="GetAddressRPRequest">
  <part name="GetAddressRPRequest" element="tns:getAddressRP" />
</message>
<message name="GetAddressRPResponse">
  <part name="GetAddressRPResponse" element="tns:getAddressRPResponse" />
</message>

<portType name="PersonMultiplePortType"
    wsrp:ResourceProperties="PersonNameResourcePropertiesSet PersonAddressResourcePropertiesSet">

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

  <operation name="SetResourceProperties">
    <input  name="SetResourcePropertiesRequest"
      message="wsrpw:SetResourcePropertiesRequest"
      wsa:Action="http://.../wsrf/2004/06/wsrf-WS-ResourceProperties/SetResourceProperties"/>
    <output name="SetResourcePropertiesResponse" message="wsrpw:SetResourcePropertiesResponse"
      wsa:Action="http://../wsrf/2004/06/wsrf-WS-ResourceProperties/SetResourcePropertiesResponse"/>
    <fault  name="ResourceUnknownFault" message="wsrpw:ResourceUnknownFault" />
    <fault  name="InvalidSetResourcePropertiesRequestContentFault"
      message="wsrpw:InvalidSetResourcePropertiesRequestContentFault" />
    <fault  name="UnableToModifyResourcePropertyFault"
      message="wsrpw:UnableToModifyResourcePropertyFault" />
    <fault  name="InvalidResourcePropertyQNameFault"
      message="wsrpw:InvalidResourcePropertyQNameFault" />
    <fault  name="SetResourcePropertyRequestFailedFault"
      message="wsrpw:SetResourcePropertyRequestFailedFault" />
  </operation>

  <operation name="Subscribe">
    <input message="wsntw:SubscribeRequest"
      wsa:Action="http://.../wsn/2004/06/wsn-WS-BaseNotification/Subscribe"/>
    <output message="wsntw:SubscribeResponse"
      wsa:Action="http://.../wsn/2004/06/wsn-WS-BaseNotification/SubscribeResponse"/>
    <fault name="TopicPathDialectUnknownFault" message="wsntw:TopicPathDialectUnknownFault"/>
```

```
      <fault name="SubscribeCreationFailedFault" message="wsntw:SubscribeCreationFailedFault"/>
      <fault name="ResourceUnknownFault" message="wsntw:ResourceUnknownFault"/>
    </operation>

    <!--  name attribute in input and output element is optional-->

    <operation name="changeName">
      <input  name="ChangeNameRequest"  message="tns:ChangeNameRequest" />
      <output name="ChangeNameResponse" message="tns:ChangeNameResponse" />
    </operation>

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

      <operation name="changeAddress">
      <input  name="ChangeAddressRequest"  message="tns:ChangeAddressRequest" />
      <output name="ChangeAddressResponse" message="tns:ChangeAddressResponse" />
    </operation>

    <operation name="getAddressRP">
      <input  name="GetAddressRPRequest"  message="tns:GetAddressRPRequest" />
      <output name="GetAddressRPResponse" message="tns:GetAddressRPResponse" />
      </operation>

  </portType>
</definitions>
```

The most important thing about this WSDL is how two resource properties set are mentioned in the single portType.

```
      <element name="PersonNameResourcePropertiesSet">
        <complexType>
          <sequence>
            <element ref="myTypes:myName"/>
          </sequence>
        </complexType>
      </element>
      <element name="PersonAddressResourcePropertiesSet">
        <complexType>
          <sequence>
            <element ref="myTypes:myAddress"/>
          </sequence>
        </complexType>
      </element>

<portType name="PersonMultiplePortType"
   wsrp:ResourceProperties="PersonNameResourcePropertiesSet PersonAddressResourcePropertiesSet">
```

Attribute "*wsrp:*ResourceProperties" in the portType element can have multiple values separated by space in single pair of double quotation marks.
I have strike out two operations GetResourceProperty and SetResourceProperties because technically they are useless in this application and I should not have them in the WSDL but as I copied the WSDL from last tutorial so keeping them here. If you decide to remove them doesn't makes any difference.
The value of name attribute in the "*definition*" element is "*PersonMultipleService*" therefore client will use "*PersonMultipleServiceAddressingLocator*" and value of name attribute in the "*portType*" element is "*PersonMultiplePortType*" therefore "*PersonMultipleServiceAddressingLocator*" will have

method with the name "*getPersonMultiplePortTypePort(..)*" which takes EndPointReference. This WSDL is once again using our existing schema by declaring the namespace in the "*definition*" section and importing the schema in the "*types*" element; below is the schema from last tutorials just for sake of finishing the discussion.

```
<?xml version="1.0" encoding="utf-8" ?>

<xs:schema targetNamespace="http://test.wsrf.resource.sharing"
      elementFormDefault="qualified" xmlns="http://test.wsrf.resource.sharing"
      xmlns:xs="http://www.w3.org/2001/X_LSchema">

      <xs:complexType  name="name">
            <xs:sequence>
                  <xs:element name="title" type="xs:string" />
                  <xs:element name="firstName" type="xs:string" />
                  <xs:element name="lastName" type="xs:string" />
                  <xs:element name="middleName" type="xs:string" />
            </xs:sequence>
      </xs:complexType >

      <xs:complexType name="address">
            <xs:sequence>
                  <xs:element name="houseNumber" type="xs:string" />
                  <xs:element name="streetName" type="xs:string" />
                  <xs:element name="city" type="xs:string" />
                  <xs:element name="county" type="xs:string" />
                  <xs:element name="country" type="xs:string" />
            </xs:sequence>
      </xs:complexType>

      <xs:complexType name="person">
            <xs:sequence>
                  <xs:element name="name" ref="myName" />
                  <xs:element name="address" ref="myAddress" />
            </xs:sequence>
      </xs:complexType>

      <xs:element name="myName" type="name"></xs:element>
      <xs:element name="myAddress" type="address"></xs:element>
      <xs:element name="myPerson" type="person"></xs:element>
</xs:schema>
```

Configuration Files

We have slight modification in our existing configuration files and it is better to discuss them before starting implementation; as implementation logic depends on the parameters declared in the configuration files; their names and their scope i.e. local or global.

Deployment Descriptor

This time our *deploy-server.wsdd* is similar to our earlier tutorials with single Instance Service "*PersonMultipleService*" and single Factory Service "*PersonMultipleFactoryService*". Factory Service has declared single parameter called "instance" which reference to the Instance Service. In the deploy-server.wsdd for Instance Service I have strike out GetProvider and SetProvider to indicate that even removing them has no impact on the application although I still have in the WSDD because for this application default implementation of GetResourceProperty and SetResourceProperties will not work for us

as this looks for the resource with the name “home” in the JNDI configuration file. Below is the complete deployment descriptor.

```
<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="PersonMultipleService" provider="Handler" use="literal" style="document">
    <parameter name="providers" value="GetRPProvider SetRPProvider SubscribeProvider"/>
    <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.wsrf.sharing.multiple.person.PersonMultipleService"/>
    <wsdlFile>share/schema/tutorial/PersonMultiple_service.wsdl</wsdlFile>
  </service>

  <!-- Factory service -->
    <service name="PersonMultipleFactoryService" provider="Handler" use="literal" style="document">
         <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="instance" value="PersonMultipleService"/>
        <parameter name="className"
             value="uk.ac.dl.wsrf.sharing.multiple.person.PersonMultipleFactoryService"/>
        <wsdlFile>share/schema/tutorial/FactoryPersonMultiple_service.wsdl</wsdlFile>
    </service>

</deployment>
```

JNDI Configuration File

This time we don’t have any global resource and we have two local resources; this choice makes no difference on the implementation of our application and more or less it is based on personal programming style. Local resources are declared for the service “*PersonMultipleService*” named “*NameHome*” and “*AddressHome*”. Our factory Service has two links to reference each of local resource declared for the “*PersonMultipleService*”; which are also named as “*NameHome*” and “*AddressHome*”. I am repeating the warning from last two tutorials that if your resource name has any name other than “home”; then you can’t use the getHome method from the Resource Context class and InitialContext should be used for lookup. This will be demonstrated in the Factory Service. Below is the complete “*deploy-jndi-config.xml*”.

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

    <service name="PersonMultipleService">
        <resource
          name="NameHome" type="uk.ac.dl.wsrf.sharing.multiple.person.PersonNameResourceHome">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>
                <parameter>
                    <name>resourceClass</name>
                <value>uk.ac.dl.wsrf.sharing.multiple.person.PersonNameResource</value>
                </parameter>
                <parameter>
                    <name>resourceKeyName</name>
                    <value>{http://sharing.wsrf.dl.ac.uk/multiple/person}PersonNameKey</value>
```

```
                </parameter>
                <parameter>
                    <name>resourceKeyType</name>
                    <value>java.lang.Integer</value>
                </parameter>
            </resourceParams>
        </resource>

        <resource
            name="AddressHome" type="uk.ac.dl.wsrf.sharing.multiple.person.PersonAddressResourceHome">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>
                <parameter>
                    <name>resourceClass</name>
                    <value>uk.ac.dl.wsrf.sharing.multiple.person.PersonAddressResource</value>
                </parameter>
                <parameter>
                    <name>resourceKeyName</name>
                    <value>{http://sharing.wsrf.dl.ac.uk/multiple/person}PersonAddressKey</value>
                </parameter>
                <parameter>
                    <name>resourceKeyType</name>
                    <value>java.lang.Integer</value>
                </parameter>
            </resourceParams>
        </resource>
    </service>

    <!-- Factory service -->
    <service name="PersonMultipleFactoryService">
      <resourceLink name="NameHome" target="java:comp/env/services/PersonMultipleService/NameHome"/>
      <resourceLink name="AddressHome" target="java:comp/env/services/PersonMultipleService/AddressHome"/>
    </service>

</jndiConfig>
```


Implementation Details

In this tutorial there are couple of extra classes; two Resource Home's and two Resources' for *"NameResource"* and *"AddressResource"*, one Factory Service, one Instance Service and one interface for QNames. Below is the table for all classes with their proper names used in this tutorial:

Class Name	Description
PersonAddressResourceHome	This is the Home class for the PersonAddressResource with single method create(); which returns the ResourceKey after successful instantiation of the PersonAddressResource.
PersonAddressResource	Implementation of our AddressResource
PersonNameResourceHome	This is class is similar in logic to PersonAddressResourceHome. This is the Home class for the PersonNameResource with single method create(); which returns the ResourceKey after successful instantiation of the PersonNameResource.
PersonNameResource	Implementation of our NameResource
PersonServiceQNames	This is utility interface with few static and final variables to wrap different QNames for ResourceProperties and ResourcePropertySet.
PersonMultipleFactoryService	This is our Factory Service to instantiate both resources and returning the

	array of EndPointReference.
PersonMultipleService	Our Instance Service which manages both resources.

Implementing Factory Service

Our Factory Service has one public createResource method in accordance with WSDL and few private utility methods to instantiate NameResource and AddressResource.

```
package uk.ac.dl.wsrf.sharing.multiple.person;

import java.rmi.RemoteException;
import java.net.URL;

import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceKey;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.apache.axis.MessageContext;
import org.globus.wsrf.utils.AddressingUtils;
import org.globus.wsrf.container.ServiceHost;
import uk.ac.dl.wsrf.sharing.multiple.person.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import org.globus.wsrf.Constants;


public class PersonMultipleFactoryService {

    /* Implementation of createResource Operation */
    public CreateResourceResponse createResource(CreateResource request) throws
            RemoteException {
        System.out.println("PersonMultipleFactoryService createResource");
        EndpointReferenceType eprArray[] = new EndpointReferenceType[2];
        EndpointReferenceType epr = null;
        ResourceKey key = null;
        key = createNameResource();
        //EPR is array in the CreateResourceResponse, so it should be initialized
        // with the size before using .. to avoid null point array exception....
        CreateResourceResponse response = new CreateResourceResponse(eprArray);
        /* We construct the instance's endpoint reference. The instance's service
         * path can be found in the WSDD file as a parameter. */
        try {
            URL baseURL = ServiceHost.getBaseURL();
            String instanceService = (String) MessageContext
                        .getCurrentContext().getService().getOption("instance");
            String instanceURI = baseURL.toString() + instanceService;
            // The endpoint reference includes the instance's URI and the resource key
            epr = AddressingUtils.createEndpointReference(instanceURI, key);
            response.setEndpointReference(0, epr);
            key = createAddressResource();
            // calling again overwirte the previous ReferenceProperties
            epr = AddressingUtils.createEndpointReference(instanceURI, key);
            response.setEndpointReference(1, epr);
        } catch (Exception e) {
            e.printStackTrace();
        }
        /* Finally, return the endpoint reference in a CreateResourceResponse */
        return response;
    }

    private ResourceKey createNameResource() {
        System.out.println("PersonMultipleFactoryService createNameResource");
```

```
        ResourceContext ctx = null;
        PersonNameResourceHome home = null;
        ResourceKey key = null;

        /* First, we create a new NameResource through the NameResourceHomeFactory  */
        try {
            Context initialContext = new InitialContext();
            ctx = ResourceContext.getResourceContext();
            String name = Constants.JNDI_SERVICES_BASE_NAME + ctx.getService() +"/NameHome";
            home = (PersonNameResourceHome) initialContext.lookup(name);
            key = home.create();
            return key;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    private ResourceKey createAddressResource() {
        System.out.println("PersonMultipleFactoryService createAddressResource");
        ResourceContext ctx = null;
        PersonAddressResourceHome home = null;
        ResourceKey key = null;

        /* First, we create a new NameResource through the NameResourceHomeFactory  */
        try {
            Context initialContext = new InitialContext();
            ctx = ResourceContext.getResourceContext();
            String name = Constants.JNDI_SERVICES_BASE_NAME + ctx.getService() +"/AddressHome";
            home = (PersonAddressResourceHome) initialContext.lookup(name);
            key = home.create();
            return key;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```


Below are the important points related to Factory Service implementation:

1. "createResource" is declaring an array of "`EndpointReferenceType`" which is used to create the object of "`CreateResourceResponse`". If you instantiate "`CreateResourceResponse`" with default constructor then set the "`EndpointReferenceType[]`" manually, instantiating the "`CreateResourceResponse`" doesn't instantiate the array, because it is developers responsibility to set the size of array, one alternative is to use `maxOccurs="2"` rather than `maxOccurs="unbounded"` in the WSDL for Factory Service.

```
EndpointReferenceType eprArray[] = new EndpointReferenceType[2];
CreateResourceResponse response = new CreateResourceResponse(eprArray);
```

2. "createResource" calls `createNameResource()and createAddressResource()`both of them returns ResourceKey. Each of the method is using InitialContext to retrieve the information about corresponding Resource before instantiating the Resource through ResourceHome.

```
try {
    Context initialContext = new InitialContext();
    ctx = ResourceContext.getResourceContext();
    String name = Constants.JNDI_SERVICES_BASE_NAME + ctx.getService() + "/NameHome";
    home = (PersonNameResourceHome) initialContext.lookup(name);
```

```
            key = home.create();
            return key;
        } catch (Exception e) {
            e.printStackTrace();
        }
```

3. "createResource" creates two EndPointReferenceType for each resource and set them in the "*CreateResourceResponse*" object before returning the "*CreateResourceResponse*" object.

```
try {
    URL baseURL = ServiceHost.getBaseURL();
    String instanceService = (String) MessageContext
                        .getCurrentContext().getService().getOption("instance");
    String instanceURI = baseURL.toString() + instanceService;
    // The endpoint reference includes the instance's URI and the resource key
    epr = AddressingUtils.createEndpointReference(instanceURI, key);
    response.setEndpointReference(0, epr);
    key = createAddressResource();
           // calling again overwirte the previous ReferenceProperties
    epr = AddressingUtils.createEndpointReference(instanceURI, key);
    response.setEndpointReference(1, epr);
} catch (Exception e) {
      e.printStackTrace();
}
```

Implementation of Resource Home

Our Resource Homes are exactly similar and they both are from previous tutorials and need no further explanation, below are the both *PersonAddressResourceHome* and *PersonNameResourceHome classes:*

PersonAddressResourceHome

```
package uk.ac.dl.wsrf.sharing.multiple.person;

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

public class PersonAddressResourceHome extends ResourceHomeImpl {
    public PersonAddressResourceHome() {
    }

    private int counter = 0;
    public ResourceKey create() {
        System.out.println("PersonAddressResourceHome create() " + counter++);
        try {
            PersonAddressResource resource = (PersonAddressResource)createNewInstance();
            resource.initialize();
            ResourceKey key = new SimpleResourceKey(keyTypeName, resource.getID());
            this.add(key, resource);
            resources.put(key, resource);
            return key;
        } catch (Exception e) {
            System.out.println("Exception when creating PersonAddressResource ");
            e.printStackTrace();
            return null;
        }
    }

}
```

PersonNameResourceHome

```
package uk.ac.dl.wsrf.sharing.multiple.person;

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

public class PersonNameResourceHome extends ResourceHomeImpl {
    //private static java.util.Hashtable resources = new java.util.Hashtable ();

    private int counter = 0;
    public ResourceKey create() {
        System.out.println("PersonNameResourceHome create() " + counter++);
        try {
            PersonNameResource resource = (PersonNameResource)createNewInstance();
            resource.initialize();
            ResourceKey key = new SimpleResourceKey(keyTypeName, resource.getID());
            this.add(key, resource);
            resources.put(key, resource);
            return key;
        } catch (Exception e) {
            System.out.println("Exception when creating PersonNameResource ");
            e.printStackTrace();
            return null;
        }
    }

}
```

Implementation of Resources

In last tutorials we had single Resource PersonResource, but in this tutorial we have split the single Resource into two Resources with the same functionality but in different classes, there is nothing new in the implementation.

PersonAddressResource

```
package uk.ac.dl.wsrf.sharing.multiple.person;

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;
import org.globus.wsrf.TopicListAccessor;

import uk.ac.dl.wsrf.sharing.multiple.person.*;
import sharing.resource.wsrf.test.*;
import org.globus.wsrf.TopicList;
import org.globus.wsrf.impl.SimpleTopicList;
import org.globus.wsrf.impl.ResourcePropertyTopic;
import org.globus.wsrf.Topic;

public class PersonAddressResource implements Resource, ResourceProperties,
        ResourceIdentifier, TopicListAccessor {

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

    /** Stores the ResourceProperties */
    private ResourcePropertySet propSet;
```

```
    /* Variable for Resource property */
    private Address address;


    /** Resource property "name". */
    private ResourceProperty addressRP;


    /** Create Topic List**/
    private TopicList topicList;

    /** initializes the HelloWorld. */
    public void initialize() throws Exception {
        System.out.println(
                "PersonAddressResource.initialize() PersonMultipleService");

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

        // create the resource property set
        this.propSet = new SimpleResourcePropertySet(PersonServiceQNames.
                ADDRESS_RESOURCE_PROPERTIES);

        // create resource properties
        this.addressRP = new SimpleResourceProperty(PersonServiceQNames.RP_Address);

        //Initialize TopicList
        this.topicList = new SimpleTopicList(this);

        // Wrap ResourceProperty to make it Topic
        //this.nameRP = new ResourcePropertyTopic(this.nameRP);
        this.addressRP = new ResourcePropertyTopic(this.addressRP);

        // Add RP in the TopicList
        this.topicList.addTopic((Topic)this.addressRP);

        this.propSet.add(this.addressRP);
        setAddress(new Address("Daresbury", "United kingdom", "Cheshire",
                               "23", "Wilson Patten"));
        this.addressRP.add(address);

    }

    public ResourcePropertySet getResourcePropertySet() {
        return propSet;    }


    public void setAddressRP(Address addressValue) {
        setAddress(addressValue);
        this.addressRP.set(0, address);
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }
```

```
    public Object getID() {
        return id;
    }

    public TopicList getTopicList() {
        return topicList;
    }

}
```

PersonNameResource

```
package uk.ac.dl.wsrf.sharing.multiple.person;

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;
import org.globus.wsrf.TopicListAccessor;

import uk.ac.dl.wsrf.sharing.multiple.person.*;
import sharing.resource.wsrf.test.*;
import org.globus.wsrf.TopicList;
import org.globus.wsrf.impl.SimpleTopicList;
import org.globus.wsrf.impl.ResourcePropertyTopic;
import org.globus.wsrf.Topic;

public class PersonNameResource implements Resource, ResourceProperties,
        ResourceIdentifier, TopicListAccessor {

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

    /** Stores the ResourceProperties */
    private ResourcePropertySet propSet;

    /* Variable for Resource property */
    private Name name;


    /** Resource property "name". */
    private ResourceProperty nameRP;


    /** Create Topic List**/
    private TopicList topicList;

    /** initializes the HelloWorld. */
    public void initialize() throws Exception {
        System.out.println(
                "PersonNameResource.initialize() PersonMultipleService");

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

        // create the resource property set
        this.propSet = new SimpleResourcePropertySet(PersonServiceQNames.
                NAME_RESOURCE_PROPERTIES);
```

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

        //Initialize TopicList
        this.topicList = new SimpleTopicList(this);

        // Wrap ResourceProperty to make it Topic
        this.nameRP = new ResourcePropertyTopic(this.nameRP);

        // Add RP in the TopicList
        this.topicList.addTopic((Topic)this.nameRP);

        this.propSet.add(this.nameRP);
        setName(new Name("Asif ", "Akram", " ", "Mr."));
        this.nameRP.add(name);
    }

    public ResourcePropertySet getResourcePropertySet() {
        return propSet;
    }

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

    public Name getName() {
        return name;
    }

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

    public Object getID() {
        return id;
    }

    public TopicList getTopicList() {
        return topicList;
    }
}
```

Implementation of Instance Service

In case of Resource we have split it into two Resources but for Instance Service we have combined the business logic of two Instance Services from last tutorials into single Instance Service. The most important thing is private operation getKey(String keyName) which uses the MessageContext to get hold of message, from that message it retrieves SOAPHeader and from SOAPHeader it gets the ResourceKey. getKey(String keyName) methods work on the passed key, and changeName() and getNameRP() methods assume that client is invoking those methods with NameResource; therefore the header will have the element "*PersonNameKey*"; similarlty chagneAddress() and getAddressRP() methods can be invoked by passing unique identification of AddressResource i.e. "*PersonAddressKey*". If clients calls wrong/invalid method with wrong EndPointRefernce then exception will be thrown. ResourceKey returned by the getKey(..) method is passed to either getAddressResource() or getNameResource() method and each of them returns the corresponding Resource.

```
package uk.ac.dl.wsrf.sharing.multiple.person;

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.sharing.multiple.person.*;
import sharing.resource.wsrf.test.*;
import javax.naming.Context;
import org.globus.wsrf.Constants;
import javax.naming.InitialContext;
import javax.xml.namespace.QName;
import org.apache.axis.MessageContext;
import javax.xml.soap.SOAPElement;
import org.globus.wsrf.impl.SimpleResourceKey;

public class PersonMultipleService {

    public PersonMultipleService() throws RemoteException { }

    public String changeName(Name name) throws RemoteException {
        System.out.println("PersonMultipleService changeName Method Called");
        PersonNameResource nameResource = getNameResource(getkey("PersonNameKey"));
        nameResource.setNameRP(name);
        return "Name Changed " + name.getFirstName() + " " + name.getLastName() +" !";
    }

    public Name getNameRP(GetNameRP params) throws RemoteException {
        System.out.println("PersonService1 getNameRP Method Called");
        PersonNameResource nameResource = getNameResource(getkey(
                "PersonNameKey"));
        return nameResource.getName();
    }

    public String changeAddress(Address address) throws RemoteException {
        System.out.println("PersonService changeAddress Method Called");
        PersonAddressResource resource = getAddressResource(getkey("PersonAddressKey"));
        resource.setAddressRP(address);
        return "Address Changed " + address.getCity() + " " + address.getCountry() + " !";
    }

    public Address getAddressRP(GetAddressRP params) throws RemoteException {
        System.out.println("PersonService getAddressRP Method Called");
        PersonAddressResource resource = getAddressResource(getkey("PersonAddressKey"));
        return resource.getAddress();
    }
    public PersonNameResource getNameResource(ResourceKey key) throws
            RemoteException {
        System.out.println("PersonMultipleService getNameResource() Method Called");
        Object resource = null;
        ResourceContext ctx = null;
        PersonNameResourceHome home = null;

        /* First, we create a new NameResource through the NameResourceHomeFactory  */
        try {
            Context initialContext = new InitialContext();
            ctx = ResourceContext.getResourceContext();
            String name = Constants.JNDI_SERVICES_BASE_NAME + ctx.getService() +"/NameHome";
            home = (PersonNameResourceHome) initialContext.lookup(name);
            resource = home.find(key);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }

        PersonNameResource nameResource = (PersonNameResource) resource;
        return nameResource;
    }

    public PersonAddressResource getAddressResource(ResourceKey key) throws
            RemoteException {
        System.out.println(
                "PersonMultipleService getAddressResource() Method Called");
        Object resource = null;
        ResourceContext ctx = null;
        PersonAddressResourceHome home = null;

        /* First, we create a new NameResource through the NameResourceHomeFactory  */
        try {
            Context initialContext = new InitialContext();
            ctx = ResourceContext.getResourceContext();
            String name = Constants.JNDI_SERVICES_BASE_NAME + ctx.getService() +"/AddressHome";
            home = (PersonAddressResourceHome) initialContext.lookup(name);
            resource = home.find(key);
        } catch (Exception e) {
            e.printStackTrace();
        }

        PersonAddressResource addressResource = (PersonAddressResource) resource;
        return addressResource;
    }

    public ResourceKey getkey(String keyName) {
        QName keyQName = null;
        Integer id = null;
        try {
            MessageContext context = MessageContext.getCurrentContext();
            java.util.Iterator iterator = context.getCurrentMessage().
                                          getSOAPHeader().getChildElements();

            while (iterator.hasNext()) {
                SOAPElement tempElement = (SOAPElement) iterator.next();
                if (tempElement.getLocalName().equals(keyName)) {
                    System.out.println(tempElement.toString());
                    System.out.println(tempElement.getFirstChild().getNodeValue());
                    id = new Integer(tempElement.getFirstChild().getNodeValue());
                    keyQName = new QName("http://sharing.wsrf.dl.ac.uk/multiple/person",keyName);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        ResourceKey key = new SimpleResourceKey(keyQName, id);
        return key;
    }
}
```

Implementation of Interface

Our Interface is simple enough with only addition of QName for additional Resource Property Set.

package uk.ac.dl.wsrf.sharing.multiple.person;

```
import javax.xml.namespace.QName;

public interface PersonServiceQNames {

    public static final String NS = "http://test.wsrf.dl.ac.uk/multiple/person";
    public static final String NS2 = "http://test.wsrf.resource.sharing";

    public static final QName RP_NAME = new QName(NS2, "myName");
    public static final QName RP_Address = new QName(NS2, "myAddress");
    public static final QName NAME_RESOURCE_PROPERTIES = new QName(NS,
            "PersonNameResourcePropertiesSet");
    public static final QName ADDRESS_RESOURCE_PROPERTIES = new QName(NS,
            "PersonAddressResourcePropertiesSet");
}
```

Implementation of Client

I have kept the client quite simple rather than making it over complicated which makes it quite fragile. In the client logic I already know the order in which two EndPointReference are returned by the Instance Service and using them accordingly later in the main method of the client but in reality this is not the case. I should have parsed the array of EndPointReference to see which one has `PersonNameKey` and which one has `PersonAddressKey` and then assign them to respective variables; the logic will be similar to something which we have used partially in the Instance Service. This time `CreateResourceResponse` has array of EndPointReference which I have assigned to "`service1InstanceEPR`".

```
private static EndpointReferenceType service1InstanceEPR[] = null;
service1InstanceEPR = createResponse.getEndpointReference();
```

EndPointReference at different index in the array is used directly to obtain respective portType. Be sure that both time the port variable is same but everytime different EndPointReference is passed. It is clients responsibility to obtain appropriate port with corresponding EndPointReference before calling respective methods on that particular resource.

```
port = locator.getPersonMultiplePortTypePort(service1InstanceEPR[0]);
port.getNameRP(new GetNameRP());
port = locator.getPersonMultiplePortTypePort(service1InstanceEPR[1]);
port.getAddressRP(new GetAddressRP());
```

Client is first passing EndPointReference for NameResource to `getPersonMultiplePortTypePort` method and then calling different methods. Client can't call getResourceProperty() method what I mean is something similar to statement below which we have used nearly in all Tutorials is not possible:

~~*GetResourcePropertyResponse response = port.getResourceProperty(PersonServiceQNames.RP_NAME);*~~

Reason are very similar to what we are talking in the last two tutorials first of all default implementation will look for any local resource for Instance Service in the JNDI Configuration file with the name "home" and we don't have any such resource; secondly even if we have any resource with the name "home"; we can't use this method for other resources managed by the same Instance Properties which means lack of consistence in the clients code and client can't see the JNDI Configuration file; so he will never realise the problem on the server side that why for one Resource getResourceProperty() is working but for other resource it is throwing excpetion. Although errors says ***"<faultcode> soapenv:Server.userException </faultcode>***", but in reality it is poor implementation on the Server Side. The best choice is to implement yourself getResourceProperty() method in the Instance Service with the logic that it checks the QName of the Resource Key in the message and calls corresponding Resource Home to find the Resource. This is not only true for the getResourceProperty() method but for all WSRF based methods related to Resource. In the

next tutorial we will implement getResourceProperty() in the Instance Service which will parse the message and return the ResourceProperty of relevant resource and that logic is not straight forwardly applicable to every Instance Service managing multiple resources; but in fact should be modified according to the requirements of the Instance Service

I have highlighted the code in different colours related to different Resources managed by the Instance Service. It seems lot of work to be done for managing multiple Resources from Single Instance Service but there is no other choice. Life Has No Short Cut.

```
package uk.ac.dl.wsrf.sharing.multiple.person.client;

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.utils.FaultHelper;

import uk.ac.dl.wsrf.sharing.multiple.person.service.*;
import uk.ac.dl.wsrf.sharing.multiple.person.*;

import javax.xml.rpc.*;

public class ClientName extends BaseClient {

    final static PersonMultipleServiceAddressingLocator locator =
            new PersonMultipleServiceAddressingLocator();

    final static FactoryPersonMultipleServiceAddressingLocator factoryLocator =
            new FactoryPersonMultipleServiceAddressingLocator();

    private static EndpointReferenceType service1InstanceEPR[] = null;

    private void getInstanceEPR() {
        try {
            PersonMultipleFactoryPortType factoryPort = factoryLocator.
                    getPersonMultipleFactoryPortTypePort(this.getEPR());
            CreateResourceResponse createResponse = factoryPort
                    .createResource(new CreateResource());
            service1InstanceEPR = createResponse.getEndpointReference();
            for (int i = 0; i < service1InstanceEPR.length; i++){
                service1InstanceEPR[i].getAddress().setPort(8082);
            }
        } catch (ServiceException ex) {
            ex.printStackTrace();
        } catch (Exception ex1) {
            ex1.printStackTrace();
        }
    }

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

        // 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 {
            client.getInstanceEPR();
            PersonMultiplePortType port;
            port = locator.getPersonMultiplePortTypePort(service1InstanceEPR[0]);
            Name tempName = port.getNameRP(new GetNameRP());
            System.out.println("Name Before Changing:  " + tempName.getTitle() +
                               " " + tempName.getFirstName() + " " +
                               tempName.getLastName());
            String result = port.changeName(new Name("AsIf", "AkRaM", "..."," Mr."));
            System.out.println(result);
            tempName = port.getNameRP(new GetNameRP());
            System.out.println("Name After Changing:  " + tempName.getTitle() +
                               " " + tempName.getFirstName() + " " +
                               tempName.getLastName());

            port = locator.getPersonMultiplePortTypePort(service1InstanceEPR[1]);
            Address tempAddress = port.getAddressRP(new GetAddressRP());
            System.out.println("Address before Changing:  " +
                               tempAddress.getHouseNumber() + " " +
                               tempAddress.getStreetName() + " " +
                               tempAddress.getCity()
                               + " " + tempAddress.getCounty() + " " +
                               tempAddress.getCountry());
            result = port.changeAddress(new Address("London", "United Kingdom",
                    "London", "38", "Hamilton Street"));
            System.out.println(result);
            tempAddress = port.getAddressRP(new GetAddressRP());
            System.out.println("Address After Changing:  " +
                               tempAddress.getHouseNumber() + " " +
                               tempAddress.getStreetName() + " " +
                               tempAddress.getCity()
                               + " " + tempAddress.getCounty() + " " +
                               tempAddress.getCountry());
        } catch (Exception e) {
            if (client.isDebugMode()) {
                FaultHelper.printStackTrace(e);
            } else {
                System.err.println("Error: " + FaultHelper.getMessage(e));
            }
        }
    }
}
```

Testing the Client

To test the client fist deploy the service using built file and run following command:

%GLOBUS_LOCATION%\bin\ClientPersonMultiple -s http://localhost:8082/wsrf/services/PersonMultipleFactoryService

Be sure to start your TCP Monitor which should be listening on the port 8082 and forwarding messaged to the port 8080. If everything goes well then you can see the following output.

Name Before Changing: Mr. Asif Akram

Name Changed AsIf AkRaM !
Name After Changing: Mr. AsIf AkRaM
Address before Changing: 23 Wilson Patten Daresbury Cheshire United kingdom
Address Changed London United Kingdom !
Address After Changing: 38 Hamilton Street London London United Kingdom

CCLRC