Web Service Wrapper For Resource Management 1.0 Component Specification

1. Design

This component provides an implementation of the RegistrationPersistence interface from the Registration Framework component. The implementation uses a web service defined in this component that wraps a ResourceManager instance from the Resource Management component. This way when a user registers using the new Registration Framework architecture, and this component is added as a registration persistence mechanism in the framework, the Resource Management component will also be alerted of the registration. The web service will be deployed in a JAX-WS environment. This component also defines a client class that mimics the API of the service, to make it easy for a user to access the web service remotely.

1.1 Design Patterns

Strategy –Different implementations of RegistrationEntitiesToResourceConverter interface can be plugged into ResourceManagerRegistrationPersistence class.

Adapter – This component implements this pattern because it adapts the entities from the Registration Framework component to work with the entities from Resource Management.

Delegate – Used by persistence implementation and by the service bean implementation.

1.2 Industry Standards

JAX-WS 2.0

EJB 3.0

JAXB2.1

1.3 Required Algorithms

1.3.1 This component will adapt the entities from the Registration Framework component to work with the entities from Resource Management. The mapping is this:

Registration Framework	**Resource Management**
ContestRole.Role.Name	Resource.ResourceRole.Name
Contest.Id	Resource.Project
User.Id	Resource.Id

All the fields of the User entity (including those from the abstract class) will be added into the Resource properties map.
The pairs added will be of the following form:
(user.fieldName,fieldValue);

"user." is a prefix that will be added to the field name so that there will be no conflict with another key from the map; regarding the values, for the fields that are not String objects use the string representation.

1.3.2 Mapping between method calls to the RegistrationPersistence interface and method calls to the ResourceManager service.

RegistrationPersistence	**ResourceManagerService**
register	updateResource
unregister	removeResource

1.3.3 How to generate client artifacts

~~This process is described at this link: https://java.sun.com/webservices/docs/2.0/jaxws/UsersGuide.html#mozTocId69398~~

The artifacts are auto generated by JBoss and This component provides a client.

1.3.4 About the Resource's properties map.

The Resource will be passed between client and server for the updateResource(), removeResource(), updateResources(), getResource() methods.

But the Resource.properties map does not have proper getter/setter nesscessary for marshall/unmarshall between the client and server.

To address this issue, this component use a CopiedResource which extends the Resource class by adding an addtition properties map, which can be successfully marshalled/unmarshalled.

The ResourceManagerServiceClient and ResourceManagerServiceBean will take care the sync of the new properties map and the original properties map.

Each time when server sends a Resource to client, or when client sends a Resource to server, they should at first convert the Resource to a CopiedResource object and then sends the CopiedResource object instead. This is done by following:

1. If Resource is null, return a null CopiedResource
2. Create a new CopiedResource
3. If Resource has id set(>0), the set the id of CopiedResource
4. Copy all other fields values from Resource into CopiedResource
5. Copy the properties map:
   CopiedResource.setProperties(Resource.getAllProperties())

Each time when server receives a Resource from client, or when client receives a Resource from server, the Resource object received is actually an instance of CopiedResource. Then the two properties map need be synced. This is down by following:

```
Map < String, Object > properties = CopiedResource.getProperties();
For each entry in the above properties map {
    CopiedResource.setProperty(key, properties.get(key));
}
```

1.4 Component Class Overview

ResourceManagerRegistrationPersistence
This class is an implementation of the RegistrationPersistence interface from the Registration Framework component. It uses a RegistrationEntitiesToResourceConverter implementation to convert the registration entities to a Resource object. It also uses a ResourceManagerServiceClient object which provides access to a service that will be used to update or remove the Resource object created with the converter.
This class is thread safe since ResourceManagerServiceClient class is thread safe and RegistrationEntitiesToResourceConverter implementations are also expected to be thread safe.

ResourceManagerService
This is a web interface that defines the contract for implementations that will allow the user to work with resources. The methods provided will allow the user to:
-update a resource, an array of resources, resource role, notification type
-remove a resource, resource role, notifications, notification type
-get a resource identified by its id
-get all resource roles, notification types
-add or get notifications
It will be annotated with @WebService(name="ResourceService")
Implementations are required to be thread safe.

ResourceManagerServiceBean
This class is an EJB3 Stateless Session Bean web service endpoint implementation of the ResourceManagerService interface.
It will be annotated with:
@WebService(endpointInterface="ccom.topcoder.registration.persistence.ResourceManagerService", serviceName = "ResourceService", targetNamespace = "http://www.topcoder.com/ResourceService")
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
This service has a field of ResourceManager type to which it will delegate in each method.

This class is thread safe because it will be deployed in an ejb container which will ensure thread safety.

ResourceManagerServiceClient
This class represents a client to a resource manager service. The proxy to the web service will be created using a url that identifies the wsdl document location; this url is provided in the constructors of this class. The methods defined in this class exactly match the methods of the ResourceManager interface.
This class is thread safe since the implementations of ResourceManagerService web service interface are expected to be thread safe.

RegistrationEntitiesToResourceConverter
This interface defines the contract for implementations that will convert registration entities to a Resource object.
Implementations are required to be thread safe.

RegistrationEntitiesToResourceConverterImpl
This class is the default implementation of the RegistrationEntitiesToResourceConverter interface. It has a method that will create a Resource object, set its fields using the parameters and then return the Resource object.
This class is thread safe because it has no state.

CopiedResource
The purpose to introduce this class is that the Resource DOES NOT have proper getter/setter for the properties map field. Thus, the properties will be lost when marshall/unmarshall between the client and server.

To follow the methods contract defined by the ResourceManagerService web service interface, this class directly extends Resource with only an additional properties map field added. And the added properties map is properly defined and can be successfully marshalled/unmarshalled when passing between the client and server.

GetResourceResponse
This class represents the response of getResource() method. The reason to introduce it is to use a field declared as CopiedResource type to override the Resource type declared by the getResource method.

UpdateResourceRequest
This class represents the request of updateResource method. The reason to introduce it is to use a field declared as CopiedResource type to override the Resource type declared by the updateResource method.

UpdateResourcesRequest
This class represents the request of updateResources method. The reason to introduce it is to use a field declared as CopiedResource type to override the Resource type declared by the updateResources method

RemoveResourceRequest
This class represents the request of removeResource method. The reason to introduce it is to use a field declared as CopiedResource type to override the Resource type declared by the removeResource method.

This class directly extends UpdateResourceRequest because the request parameters of removeResource() and updateResource() are essentially same from the aspect of web service.

1.5 Component Exception Definitions

WebServiceWrapperForResourceManagementException
This custom exception is the base exception of all custom checked exceptions defined in this component. It will not be directly thrown by any class.

RegistrationEntitiesToResourceConversionException
This is a custom exception that will be thrown by RegistrationEntitiesToResourceConverter implementations if a conversion error occurs. It will be thrown by RegistrationEntitiesToResourceConverter implementations and in the register and unregister methods from the ResourceManagerRegistrationPersistence class.

ConfigurationException
This is a custom exception that will be thrown to indicate configuration errors. It will be thrown in the constructor , that takes a ConfigurationObject instance, of ResourceManagerRegistrationPersistence class.

ResourceManagementException
This is a custom exception that will be thrown by the RegistrationManagerService implementations and ResourceManagerServiceClient class (thrown in all methods) if an error occurs when managing resources.

This exception is annotated with “@WebFault(
```
        name = "resourceManagementFault",
        faultBean =
"com.topcoder.registration.persistence.ResourceManagementFault"
        )”
```
To work around an issue about BaseException

ResourceManagementFault
This class represents the fault bean of the related ResourceManagementException exception. The purpose to introduce it is to avoid an issue about JBossWS and Base Exception component.
The BaseException is not suitable to be used with JBossWS, because it has a getter getInformation() but there is no corresponding information field.
By introducing the fault bean, the JBossWS will instead use the fault bean to marshall/unmarshall the exception and thus the problem about BaseException is avoided.

ResourceManagerServiceClientCreationException
This is a custom exception that will be thrown by the ResourceManagerServiceClient constructors if errors occur when creating the client (when creating a ContestRegistrationService instance or when creating a proxy). It will also be thrown from the constructor that takes a configuration argument, from ResourceManagerRegistrationPersistence class.

ResourceManagerBeanInitializationException
This is a custom runtime exception that will be thrown by the ResourceManagerBean class in the initialize method, if errors occur while creating the ResourceManager instance or if file or namespace or resourceManagerKey are empty strings or if namespace or resourceManagerKey are null.

1.6 Thread Safety

The component is thread safe. ResourceManagerServiceBean is thread safe because it will be deployed in an ejb container which will ensure thread safety. ResourceManagerServiceClient class is thread safe since the implementations of RegistrationService web service interface are expected to be thread safe. ResourceManagerRegistrationPersistence is thread safe since ResourceManagerServiceClient class is thread safe and RegistrationEntitiesToResourceConverter implementations are also expected to be thread safe. RegistrationEntitiesToResourceConverterImpl is thread safe because it has no state.

2. Environment Requirements

2.1 Environment

• Development language: Java1.5
• Compile target: Java1.5 and Java1.6

2.2 TopCoder Software Components

- **Base Exception 2.0 –** to provide the base class for the custom exceptions
- **Registration Framework 1.0 –** some entities from this component are adapted to a Resource entity
- **Configuration Persistence 1.0 –** used to retrieve the ConfigurationObject instance used with Object Factory component to create a ResourceManager implementation instance
- **Configuration API 1.0 –** used for configuration purposes
- **Object Factory 2.1.0 –** used to create objects (ResourceManager and RegistrationEntitiesToResourceConverter implementations)
- **Object Factory Configuration API Plugin 1.0 –** allows creation of objects from a ConfigurationObject instance (a ConfigurationObjectSpecificationFactory will be created from a ConfigurationObject instance and then used to create a ObjectFactory instance)

- **Resource Management 1.1 –** ResourceManager interface is defined in this component; the entities from this component are also used

  NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.registration.persistence
com.topcoder.registration.service.client
com.topcoder.registration.persistence.ejbservice
com.topcoder.registration.persistence.converter

3.2 Configuration Parameters

The following parameters will need to be configured in the deployment descriptor:

Parameter	**Description**	**Values**
file	Indicates the file that contains the configuration necessary to create a ResourceManager implementation instance. **Optional.**	non empty string
namespace	Indicates the namespace from a file that contains the configuration necessary to create a ResourceManager implementation instance. **Required.**	non empty string
resourceManagerKey	The key used with ObjectFactory to create a ResourceManager implementation instance. **Required.**	non empty string
cacheResourceRoles	Indicates whether the resource roles should be cached or not. **Optional.**	“true” or “false”
cacheNotificationTypes	Indicates whether the notification types should be cached or not. **Optional.**	“true” or “false”

Configuration parameters for ResourceManagerRegistrationPersistence

Parameter	**Description**	**Values**

url	The url for the wsdl document location. **Required.**	non empty string
object_factory_config_child_name	Indicates under what name the child, containing the configuration necessary for ObjectFactory, is registered. **Required.**	non empty string
converter_key	The key used with ObjectFactory to create a RegistrationEntitiesToResourceConverter implementation instance. **Required.**	non empty string

3.3 Dependencies Configuration

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration. and README
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

See demo.

4.3 Demo

An example of the deployment descriptor is provided in ejb_jar.xml file.

1.3.1 How to create a client

```
String url = "http://localhost:8080/registration/persistence/
                    services/ResourceManagerService?wsdl";
//create a client
ResourceManagerServiceClient client = new ResourceManagerServiceClient
                                                             (url);
//or create a client with URL
 client = new ResourceManagerServiceClient(new URL(url));
//or create a client with URL and QName
client = new ResourceManagerServiceClient(new URL(url),
       new QName("http://www.topcoder.com/ResourceService",
                 "ResourceService"));
```

1.3.2 How to create a ResourceManagerRegistrationPersistence instance

```
//create a RegistrationEntitiesToResourceConverter implementation instance
RegistrationEntitiesToResourceConverter converter = new
                    RegistrationEntitiesToResourceConverterImpl();

//create a persistence instance using the created client and converter objects
RegistrationPersistence persistence = new
            ResourceManagerRegistrationPersistence(client,converter);
```

//create a persistence instance from a ConfigurationObject instance
```
RegistrationPersistence persistence1 = new
                 ResourceManagerRegistrationPersistence(configuration);
```

//register user
```
persistence.register(contest, user, contestRole);
```

//unregister user
```
persistence.unregister(contest, user, contestRole);
```

1.3.3 Methods that can be called using the client

1.3.3.1 Methods to work with Resource instances

//update a resource
```
client.updateResource(resource,”John”);
```

//remove a resource
```
client.removeResource(resource,”John”);
```

//update resources for a project
```
client.updateResources(resources, 1, “John”);
```

//get a resource by id
```
Resource resource = client.getResource(10);
```

1.3.3.2 Methods to work with ResourceRole instances

//update a resource role
```
client.updateResourceRole(resourceRole,”John”);
```

//remove a resource role
```
client.removeResourceRole(resourceRole,”John”);
```

//get all resource roles
```
ResourceRole[] resourceRoles1 = client.getAllResourceRoles() ;
```

1.3.3.3 Methods to work with Notification instances

//add notifications, of a given type, for users for a project
```
client.addNotifications(usersArray, 2, 2,”John”);
```

//remove notifications, of a given type, from users for a project
```
client.removeNotifications(usersArray, 2, 2,”John”);
```

//get users ids for all notifications for the given project and type
```
long[] users = client.getNotifications(2, 3);
```

1.3.3.4 Methods to work with NotificationType instances

//update a notification type
```
client.updateNotificationType(notificationType,”John”);
```

//remove a notification type
```
client.removeNotificationType(notificationType,”John”);
```

//get all notification types
```
NotificationType[] notificationTypes1 =
                                  client.getAllNotificationTypes() ;
```

5. Future Enhancements

New implementations of RegistrationEntitiesToResourceConverter interface could be provided.