Prerequisite Service 1.0 Component Specification

1. Design

This component which allows to user get prerequisite documents which should be signed for a specific role; get concrete prerequisite document; get all prerequisite documents; record answer for document. It defines web service interface and provides its EJB endpoint implementation. Component uses TopCoder Prerequisite Document Manager for providing all operations with persistence. It should not access to persistence by itself. The design provides the interface depicted and the EJB implementation. To create the interface, firstly I updated the WSDL with the changes discussed in forum and other necessary changes. Then I generated the interface and all related classes and exception using *wsconsume* tool of Jboss, to make the interface consistent with WSDL. Note that with JAX-WS and Jboss and WSDL is not necessary anymore because the WSDL will be generated by the endpoint interface.

At this point I refactorized the classes changing the names (with the same value in the annotations) and other changes: these changes are permitted and don't delete the functionality of the classes.

The component strongly uses the annotations in all levels: methods and classes. The classes with "Request" and "Response" suffixes represent the request and response of web services. The exceptions use the fault bean to make the exceptions available to the client side. The identity of the PrerequisiteServiceBean is set using the annotations.

1.1 Design Patterns

1.1.1 Strategy

PrerequisiteServiceBean uses strategically the DocumentManager implementation.

1.2 Industry Standards

? EJB (specifically, to ensure this component is compatible with EJB restrictions, as defined in http://java.sun.com/blueprints/qanda/ejb_tier/restrictions.html).

1.3 Required Algorithms

1.3.1 Logging

Almost all methods in PrerequisiteServiceBean (init included and except the getters) should be logged in the following manner:
? Method entrance and exit at INFO level.
? All exceptions and errors at ERROR level. This includes illegal arguments.
? Any additional logging is at the developer's discretion.

The logging is optional, therefore if the logName is null then don't provide the logging.

1.3.2 Exception creation in the PrerequisiteServiceBean

The exceptions in the PrerequisiteServiceBean are constructed using the constructors with the fault beans (the classes with "fault" suffix). Firstly create the fault bean using factory , then set the necessary information to the fault bean. At this point construct the exception with the fault bean.

This mechanism is necessary to create correctly the SOAP fault message.

The fault beans should implement Serializable because they are possiblly serialized as java objects by EJB invoke, they are also serialized in the SOAP environment.

1.3.3 Check the role and get the member id

The role checking of the user is made in almost all methods. This is the algorithm:
● get the Principal from SessionContext and cat to UserProfilePrincipal
● get the Profile map from UserProfilePrincipal
● the keys are the role ids and the values are the role names
● check if the role id is in the attributes (the keys are Long instances): if it is not present the throw the related exception
● The member id is retrieved from UserProfilePrincipal using the related method to get the user id.

1.3.4 Arguments checking in the beans

The arguments checking in all bean classes is not performed, it is performed only in the PrerequisiteServiceBean. In this mode the user can set all the possible values in the beans. The request and response classes are not used directly from this component, they are used automatically by the container.

1.4 Component Class Overview

1.4.1 com.topcoder.service.prerequisite

This package holds all the main interfaces and classes in this component.

PrerequisiteService
The pre-requisite service provides the ability to retrieve required prerequisite documents, and to record a users response to the prerequisite. This interface defines the web services endpoint with the operation. It uses the annotations to defined the requests and responses definitions.

PrerequisiteDocument
It represents the PrerequisiteDocument entity (bean) used by the service. It contains the information of a prerequisite document. The information can be null or can be empty, therefore this check is not present in the setters. It is similar and related to DocumentVersion if prerequisite manager.

PrerequisiteServiceBean

This is the EJB implementation of the PrerequisiteService interface web service endpoint. It uses an instance of DocumentManager (injected as EJB) to perform the logic of the methods. The web services annotations are presents in the endpoint interface, this bean contains only the annotation to connect the endpoint interface and this implementation. The security is provided programmatically and also with annotations. The exceptions are constructed using the fault bean because the fault message can be consumed as SOAP message, this is necessary because it's a web services. This implementations is designed to be used by the related interface and also by a different web services client: all the response, request and exceptions are automatically transformed to SOAP element.

PrerequisiteElementsFactory
This object contains factory methods for each Java content interface and Java element interface generated in the com.topcoder.service.prerequisite.ejb package. It is used by the PrerequisiteServiceBean to construct the soap elements.
A PrerequisiteElementsFactory allows you to programmatically construct new instances of the Java representation for XML content. The Java representation of XML content can consist of schema derived interfaces and classes representing the binding of schema type definitions, element declarations and model groups. Factory methods for each of these are provided in this class.

PersistenceFault
This class represents the "persistence_fault" element in WSDL. It's contained in the related exception-message class.

DocumentNotFoundFault
This class represents the "document_not_found_fault" element in WSDL. It's contained in the related exception-message class. It is not used directly, it is used by the container.

IllegalArgumentWSFault
This class represents the "illegal_argument_fault" element in WSDL. It's contained in the related with illegal argument message class.

CompetitionNotFoundFault
This class represents the "competition_not_found_fault" element in WSDL. It's contained in the related exception-message class, it's the fault bean.

RoleNotFoundFault
This class represents the "role_not_found_fault" element in WSDL. It's contained in the related exception-message class, it's the fault bean.

UserNotAuthorizedFault

This class represents the "user_not_authorized_fault" element in WSDL. It's contained in the related exception-message class.

UserAlreadyAnsweredDocumentFault

This class represents the "user_already_answered_document_fault" element in WSDL. It's contained in the related exception-message class, it's the fault bean.

RecordMemberAnswerRequest

This class represents the "recordMemberAnswer" element in WSDL. It is used by the web services request to handle the arguments of the related method.

GetAllPrerequisiteDocumentsRequest

This class represents the "getAllPrerequisiteDocuments" element in WSDL. It is used by the web services request. It has not attributes since the method has no arguments.

GetAllPrerequisiteDocumentsResponse

This class represents the "getAllPrerequisiteDocumentsResponse" element in WSDL. It is used to wrap the response of the web services and it contains the result of getAllPrerequisiteDocuments operation.

GetPrerequisiteDocumentRequest

This class represents the "getPrerequisiteDocument" element in WSDL. It is used by the web services request to handle the arguments of the related method.

GetPrerequisiteDocumentResponse

This class represents the "getPrerequisiteDocumentResponse" element in WSDL. It is used to wrap the response of the web services and it contains the result of getPrerequisiteDocument operation.

GetPrerequisiteDocumentsRequest

This class represents the "getPrerequisiteDocuments" element in WSDL. It is used by the web services request of the related operation..

GetPrerequisiteDocumentsResponse

This class represents the "getPrerequisiteDocumentsResponse" element in WSDL. It is used to wrap the response of the web services and it contains the result of getPrerequisiteDocuments operation.

RecordMemberAnswerResponse

This class represents the "recordMemberAnswerResponse" element in WSDL. It is used to wrap the response of the web services and it contains the result of getAllPrerequisiteDocuments operation. It dosn't contain attributes because the related operation in the service is void.

package-info

Annotations that can be applied to the package element are referred to as package-level annotations. An annotation with ElementType.PACKAGE as one of its targets is a package-level annotation. Package-level annotations are placed in a package-info.java file. This file should contain only the package statement, preceded by annotations. When the compiler encounters package-info.java file, it will create a synthetic interface, package-name.package-info. The interface is called synthetic because it is introduced by the compiler and does not have a corresponding construct in the source code.

This file is not a class, it contains only:

@javax.xml.bind.annotation.XmlSchema(namespace="http://www.example.org/PrerequisiteService/")

package com.topcoder.service.prerequisite.ejb;

It is uses to define the XMLSchema of the elements of this package.

1.5 Component Exception Definitions

PrerequisiteServiceException

This exception extends the BaseCriticalException. This class should be the parent exception of (almost) all exceptions thrown by PrerequisiteService..

PersistenceException

This exception is thrown automatically by the web service interface when an error occurs in the persistence layer. It wraps the generic persistence exceptions (not specific thrown by the interface of this component) thrown by the DocumentManagerBean used in the ejb implementation. It is related with "persistence_faultMsg" fault in WSDL, but it contains only annotations about "persistence_fault" because the messages are automatically constructed.

UserAlreadyAnsweredDocumentException

This exception is thrown automatically by the web service interface when the user has already answered to the document.. It is related with "user_already_answered_document_faultMsg" fault in WSDL, but it contains only annotations about "user_already_answered_document_fault" because the messages are automatically constructed.

RoleNotFoundException

This exception is thrown automatically by the web service interface when a role is not found. It is related with "role_not_found_faultMsg" fault in WSDL.

CompetitionNotFoundException

This exception is thrown automatically by the web service interface when a competition is not found. It is related with "competition_not_found_faultMsg" fault in WSDL, but it

contains only annotations about "competition_not_found_fault" because the messages are automatically constructed.

UserNotAuthorizedException
This exception is thrown automatically by the web service interface when the user is not authorized to perform the operation. It is related with "user_not_authorized_faultMsg" fault in WSDL, but it contains only annotations about "user_not_authorized_fault" because the messages are automatically constructed.

IllegalArgumentWSException
This exception is thrown automatically by the web service interface when the arguments are illegal. It is related with "illegal_argument_faultMsg" fault in WSDL, but it contains only annotations about "illegal_argument_fault" because the messages are automatically constructed. this exception is necessary because it contains the information of the illegal arguments in SOAP mode. The web service can be called also from other types of clients, different from java. It's necessary to store the information using the fault beans like other exception. WS is WebServices.

DocumentNotFoundException
This exception is thrown automatically by the web service interface when a document is not found. It is related with "document_not_found_faultMsg" fault in WSDL, but it contains only annotations about "document_not_found_fault" because the messages are automatically constructed.

1.6 Thread Safety

The thread safety is an important aspect since this component is a set web services. The Java beans are not thread safe, but it's not required that they would be thread safe because they are used within a single thread. The PrerequisiteServiceBean is thread safe because it uses an instance of DocumentManager and the implementations of this interface are required to be thread safe: the existent implementation of this interface is thread safe because it uses the transaction mechanism.
Therefore the component is completely thread safe.

2. Environment Requirements

2.1 Environment
? Development language: Java 1.5
? Compile target: Java 1.5

2.2 TopCoder Software Components
· Base Exception 2.0
  o Topcoder standard for all custom exceptions.
· Logging Wrapper 2.0
  o Used for logging the operations.

· Prerequisite Document Manager 1.0

> o is used in the PrerequisiteServiceBean to delegate the work to persistence layer

· User Group Manager 1.0.1

> o defines the Profile interfaces used by this design.

· JBoss Login Module 2.0

> o defines the UIserProfilePrincipal class used in this design

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

- J2EE 5, EJB 3.0 and JAX-WS
- JBoss: wsconsume tool was used to generated the classes, but it is not directly used by this component, it will be used for deployment

3. Installation and Configuration

3.1 Package Names

com.topcoder.service.prerequisite

com.topcoder.service.prerequisite.ejb

3.2 Configuration Parameters

3.2.1 PrerequisiteServiceBean

These parameters must be set in the configuration of the EJB (see the demo):

Parameter	Description	Details
logName	Name of the log used to retrieve the Log with LogManager Optional, the logging is optional	A valid log name, not null, not empty
documentManager	ejb-ref/ejb-local-ref definition, it is for retrieve the local or remote interface of document manager.	a valid definition of ejb-ref or ejb-local-ref entry.

3.3 Dependencies Configuration

3.3.1 TopCoder dependencies

All the dependencies are to be configured according to their component specifications.

4. Usage Notes

4.1 Required steps to test the component

? Extract the component distribution.

? Follow the instructions.

Setup

Database

1. install Informix and make sure the database is started.
2. create a database named 'document' in Informix.
3. use 'dbaccess project <path>/test_files/create.sql' to create database. it will create tables and sequences.
    Note, you can use 'dbaccess project <path>/test_files/drop.sql' to drop tables and sequences.
4. change the ip address and port in test_files/informix-ds.xml and test_files/
    InformixDBConnectionFactory.xml.

Build the prerequsite document manager ejb module.

1. go to directory test_files/Prerequisite_Document_Manager_1.0_dev_dist.
2. run 'ant package.ejb', the ejb module will be present at
    test_files\Prerequisite_Document_Manager_1.0_dev_dist\build\dist\application\ejb
3. actually, you don't need to do this step, a already built version is put in lib directory.

JBoss

1. Download and install JBoss 4.2.2.
2. copy test_files\lib\ to <jboss-home>\server\default\lib
3. modify <jboss-home\server\default\conf\login-config.xml, adding

```
<application-policy name="JBossLoginModule">
<authentication>
<login-module code="com.topcoder.security.auth.module.JBossLoginModule"
flag="required">
<module-option name="fileName">C:/demoConfigs.properties</module-option>
<module-option name="password-stacking">useFirstPass</module-option>
</login-module>
</authentication>
</application-policy>
```

Notes, fileName property should be point to test_files/demoConfigs.properties file.

4. change demoConfigs.properties, using the correct path for JBossLoginModule.xml.

Build.xml

1. there are some settings need to be adjusted according to your environment.
like
1.1 "configuration for checkstyles"
1.2 "codertura task definitation"
1.3 <property name="jboss_home" value="D:/Servers/AppServers/jboss-4.2.2.GA" />

Now, all the setup is ready. Let's start.

1. start Jboss 4.2.2
2. use 'ant deployEAR', to package ear file and deploy to jboss, a latest built is alredy saved as test_files/ prerequisite_service.ear.
3. use 'ant test' to run the test cases.
4. 'ant undeployEAR' is used to delete the ear file in JBoss. and 'ant package.ejb' is used to package the ejb jar.

? Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

4.2.1 please check the JBoss Login Module 1.0, to configure the jboss's login-config.xml properly.

4.2.2 to use this component as a ejb module.

First, configure the ejb (modify test_files\ejb\ejb-jar.xml and jboss.xml), then use 'ant package.ejb' to generate the jar module.

4.2.3 to use this component as j2ee application.

See 4.1. use 'ant deployEAR', to package ear file and deploy to jboss, a latest built is alredy saved as test_files/ prerequisite_service.ear.

4.3 Demo

4.3.1 Setup

This is the ejb-jar.xml for the server side. It defines the EJBs in the container (it defines also the PrerequisiteServiceBean used by PrerequisiteServiceBean):

```xml
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/j2ee/ejb-jar_3_0.xsd"
version="3.0">

<!-- This is the EJB declaration. It's minimal because the component
uses the annotations to declare
home and local interface -->
<enterprise-beans>
<session>
<ejb-name>DocumentManagerBean</ejb-name>
<ejb-class>
com.topcoder.service.prerequisite.document.ejb.DocumentManagerBean
</ejb-class>
<env-entry>
<env-entry-name>loggerName</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>document_manager</env-entry-value>
</env-entry>
<env-entry>
<env-entry-name>persistenceUnitName</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>document_manager</env-entry-value>
</env-entry>
<env-entry>
<env-entry-name>allowedRoleNames</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>admin,manager,Administrator</env-entry-value>
</env-entry>
</session>

<session>
<ejb-name>PrerequisiteServiceBean</ejb-name>
<ejb-class>com.topcoder.service.prerequisite.ejb.PrerequisiteServiceBean</ejb-class>
<env-entry>
<env-entry-name>loggerName</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>prerequisite_log</env-entry-value>
</env-entry>
<!-- following configuration does not works, jboss.xml is used
instead.-->
<!--
```

```
<ejb-ref>
<ejb-ref-name>documentManager</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<ejb-link>prerequisite_document_manager.jar#DocumentManagerBean</ejb-link>
</ejb-ref>
-->
</session>
</enterprise-beans>
</ejb-jar>

jboss.xml:
<?xml version="1.0"?>
<jboss>
<enterprise-beans>
<session>
<ejb-name>PrerequisiteServiceBean</ejb-name>
<jndi-name>remote/PrerequisiteService</jndi-name>
<ejb-ref>
<ejb-ref-name>documentManager</ejb-ref-name>
<jndi-name>remote/DocumentManager</jndi-name>
</ejb-ref>
</session>
</enterprise-beans>

<security-domain>java:/jaas/JBossLoginModule</security-domain>
</jboss>
```

In the server side there is no need of other configuration: web service.xml and the JAX-RPC are not required because the annotations provide automatically these information.
About the deployment in JBoss, you can see here: http://jbws.dyndns.org/mediawiki/index.php?title=Quick_Start#Consuming_web_services

"Simply wrap up the service implementation class, the endpoint interface and any custom data types in a JAR and drop them in the deploy directory. No additional deployment descriptors required. Any meta data required for the deployment of the actual web service is taken from the annotations provided on the implementation class and the service endpoint interface. JBossWS will intercept that EJB3 deployment (the bean will also be there) and create an HTTP endpoint at deploy-time: "

4.3.2 Usage

In this demo, before we show the initial data.
We have:

- 2 document with document ids 10 and 20
- 2 version of documents for each document : version 1 and 2, the content of version 2 of document 10 contains "final version of production"
- 2 competition document for each document version: one competition document is associated with competition id =20 and role id of "designer"

This is the demo:

```java
// retrieve the web service
// we can successfully retrieve the instance, but we can not call any
operations because of authentication
// problem.
Service service = Service.create(new URL(
"http://127.0.0.1:8080/prerequisite_service-prerequisite_service/
PrerequisiteServiceBean?wsdl"),
new QName("http://ejb.prerequisite.service.topcoder.com/",
"PrerequisiteServiceBeanService"));
prerequisiteService = service.getPort(PrerequisiteService.class);

Properties env = new Properties();
env.setProperty(Context.SECURITY_PRINCIPAL, "admin");
env.setProperty(Context.SECURITY_CREDENTIALS, "password");
env.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.security.jndi.JndiLoginInitialContextFactory");
InitialContext initCtx = new InitialContext(env);
// retrieve the service by ejb.
prerequisiteService = (PrerequisiteService) initCtx.lookup("remote/
PrerequisiteService");

// get all documents, only administrator can do this operation.
List<PrerequisiteDocument> allDocuments = prerequisiteService.getAllPrerequisiteDocuments();
// the list now contains the 2 documents, and using the entities of
prerequisite
// manager you can get all the sub-entities: version, competition documents
etc..

// get the prerequisite document (similar to document version) of document
with
// id=10 and version=2
PrerequisiteDocument prerequisiteDocument = prerequisiteService.getPrerequisiteDocument(10,
2);
// get the content of document versions
String content = prerequisiteDocument.getContent();
```

```java
// get the document id
long documentId = prerequisiteDocument.getDocumentId();

// get the name of document
String name = prerequisiteDocument.getName();

// get the version
int version = prerequisiteDocument.getVersion();

// get the timestamp of document version date
XMLGregorianCalendar calendar = prerequisiteDocument.getVersionDate();
// for example get the day
int day = calendar.getDay();

// get the prerequisite documents of competition 20 and role id = 1, it is
the role
// of "designer"
List<PrerequisiteDocument> prerequisiteDocuments =
prerequisiteService.getPrerequisiteDocuments(20, 1);
// it returns a single prerequisite document (document version) associated
with this
// competition and role

// create the timestamp and set to 27 February 2008
GregorianCalendar cal = new GregorianCalendar();
cal.set(Calendar.YEAR, 2008);
cal.set(Calendar.MONTH, 1);
cal.set(Calendar.DAY_OF_MONTH, 27);

// create the XMLGregorianCalendar
XMLGregorianCalendar xmlCal =
DatatypeFactory.newInstance().newXMLGregorianCalendar();

// record the member answer for the competition id = 20, the date previous
created,
// the prerequisite document retrieved and role of designer: role id =1
prerequisiteService.recordMemberAnswer(20, xmlCal, true,
prerequisiteDocument, 1);
// now a MemberAnswer entity is saved into persistence layer and the previous
// prerequisite document has an answer "true" for the competition id = 20,
and the member
// id = 17, the member id is the id user id of the caller (in
UserProfilePrincipal)
```

5. Future Enhancements

Other methods can be added to the service.