Confluence Services 1.0 Component Specification

1. Design

This component provides a wrapper to the Confluence Management component, exposing the main management functionality as web services. This component also provides client classes that will allow for easy remote access to the service methods. In both cases, JAX-WS is used.

The web service is an EJB that simply wraps a ConfluenceManager instance. To efficiently propagate exceptions, and to deal with the issue of Base Exceptions not being compatible with JBoss, a parallel set of Exceptions is provided to be used by the bean.

The client classes are provided to allow a user to call the above web service, and they come in two flavors. The first flavor is a javax.xml.ws.Service extension that allows the user to obtain the web service proxy. The second flavor, which is provided in addition to the required functionality, provides a ConfluenceManager delegate that allows a user to use the web service client transparently.

The component makes use of the Configuration API and Persistence components to obtain configuration information. It uses the Object Factory and its Configuration API-backed specification factory to instantiate objects, and the Log Manager to perform logging of activity and errors.

Since the ConfluenceManager uses overloaded methods, the web method annotations are set to provide unique names for the WSDL generation.

1.1 Design Patterns

1.1.1 Strategy

This component provides a strategy realization of the ConfluenceManager interface in the shape of ConfluenceManagerWebServiceDelegate. The ConfluenceManager is also used as a strategy by the ConfluenceManagementServiceBean

It also uses strategy with the ConfluenceManagementService so it can be used transparently by web service clients.

1.1.2 Business Interface

ConfluenceManagementService is an interface that defines the business methods in the local and remote interfaces and the session bean ConfluenceManagementServiceBean.

1.1.3 Proxy

In this design, the ConfluenceManagementService also acts as a proxy to the web service EJB.

1.1.4 *Business Delegate*

ConfluenceManagerWebServiceDelegate is a business delegate that hides the user from the fact he/she is connecting to a web service. ConfluenceManagerWebServiceDelegate uses the ConfluenceManagementServiceClient as the lookup service for the ConfluenceManagementService proxy.

1.1.5 *Factory*

ConfluenceManagementServiceClient is a factory of ConfluenceManagementService proxy instances.

1.2 Industry Standards

- Web Services (JAX-WS 2.0)
- Inversion of Control (IoC)
- EJB 3.0

1.3 Required Algorithms

1.3.1 *Logging standard for all delegate and bean methods*

This section will state the complete scenarios for logging in all delegate and bean public methods, if logging is turned on.

- Method entrance and exit will be logged with DEBUG level.
  - Entrance format: [Entering method {*className.methodName}*]
  - Exit format: [Exiting method {*className.methodName}]*. Only do this if there are no exceptions.
- Method request and response parameters will be logged with DEBUG level
  - Format for request parameters: [Input parameters[{request_parameter_name_1}:{ request_parameter_value_1}, {request_parameter_name_2}:{ request_parameter_value_2}, etc.)]]
  - Format for the response: [Output parameter {response_value}] . Only do this if there are no exceptions and the return value is not void.
  - If a request or response parameter is complex, use its toString() method. If that is not implemented, then print its value using the same kind of name:value notation as above. If a child parameter is also complex, repeat this process recursively.
- All exceptions will be logged at WARNING level. Use method log.(Level, Throwable, String). This should automatically log inner exceptions as well.
  - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry
3. Log method input parameters
4. If error occurs, log it and skip to step7

5. Log method exit
6. If not void, log method output value
7. Method exit

1.3.2 *Mapping between manager and service exceptions*

The mapping is direct between namesake exceptions, regardless of the inheritance hierarchy:

- `ConfluenceConnectionException ⇔ ConfluenceManagementServiceConnectionException`
- `ConfluenceAuthenticationFailedException ⇔ ConfluenceManagementServiceAuthenticationFailedException`
- `ConfluenceNotAuthorizedException ⇔ ConfluenceManagementServiceNotAuthorizedException`
- `ConfluenceManagerException ⇔ ConfluenceManagementServiceException`

Unfortunately, there is no process for mapping the data in the ExceptionData or the cause of the error, as it to may be a Base Exception, but the goal here is to be able to pass along the basic message and type of the exception, as we expect that logging of these error details in the EJB will suffice.

As such, the basic approach to map from a manager exception to a service exception, and vice versa, is to create an exception from the message content of the other exception.

1.4 Component Class Overview

ConfluenceManagementService
The business interface that defines all methods for accessing the Confluence Management component. It basically has the same API as the ConfluenceManager interface, and is meant to be used as a pass-through.

ConfluenceManagementServiceLocal
The local EJB interface that simply extends the ConfluenceManagementService interface, with no additional facilities. This interface should be marked with @Local annotation representing it's a local interface.

ConfluenceManagementServiceRemote
The remote EJB interface that simply extends the ConfluenceManagementService interface, with no additional facilities. This interface should be marked with @Remote annotation representing it's a remote interface.

ConfluenceManagementServiceBean
The stateless session bean that implements all operations in the ConfluenceManagementService to expose the ConfluenceManager methods as web services. All its methods are exposed, and all methods are just delegated to.

Any exceptions in the manager are mapped to the service exceptions as per CS 1.3.2.

It uses the Configuration API and Persistence components to obtain configuration information. It uses the Object Factory and its Configuration API-backed specification factory to instantiate the ConfluenceManager, and the Log Manager to perform logging of activity and errors.

ConfluenceManagementServiceClient
The service client class that is used to obtain the ConfluenceManagementService proxy. In effect it is a factory of ConfluenceManagementService instances.

ConfluenceManagerWebServiceDelegate
A simple business delegate implementation of the ConfluenceManager that uses the ConfluenceManagementServiceClient as the lookup service for the ConfluenceManagementService proxies to be used to communicate with the Web Services. All methods are implemented and all methods just delegate to the proxy.

Any exceptions in the proxy are mapped to the Confluence Management exceptions as per CS 1.3.2.

It uses the Configuration API and Persistence components to obtain configuration information. It uses the Object Factory and its Configuration API-backed specification factory to instantiate the service client, and the Log Manager to perform logging of activity and errors.

1.5 Component Exception Definitions

This component defines four new exceptions.

ConfluenceManagementServiceException
This exception is the base exception for all exceptions raised in ConfluenceManagementService. It is a direct wrapper for the Confluence Management's ConfluenceManagerException. It extends Exception

ConfluenceManagementServiceConnectionException
This exception signals an issue when an attempt to reestablish the connection to Confluence fails. It is a direct wrapper for the Confluence Management's ConfluenceConnectionException. It extends ConfluenceManagementServiceException

ConfluenceManagementServiceAuthenticationFailedException
This exception signals an issue if the authorization process fails, such as incorrect credentials being provided to be able to log in. It is a direct wrapper for the Confluence Management's ConfluenceAuthenticationFailedException. It extends ConfluenceManagementServiceException

ConfluenceManagementServiceNotAuthorizedException

This exception signals an issue if a user attempts an unauthorized call (the token used to authenticate is invalid or the user does not have access to the specific areas being manipulated). Also this exception might be thrown if the user or password is invalid. It is a direct wrapper for the Confluence Management's ConfluenceNotAuthorizedException. It extends ConfluenceManagementServiceException

ConfluenceManagementServiceConfigurationException
This exception signals an issue if the configured value is invalid. It is used in the initialize method of the EJB. It extends RuntimeException.

This exception is not mapped with the Confluence Management’s corresponding configuration exception.

1.6 Thread Safety

The underlying confluence management component is effectively thread-safe. As discussed there, the component is expected to be used in a manner that is thread-safe, and this component does do so. The bean will have its state set during initialization, and not modified afterwards, which makes it effectively thread-safe. Otherwise, both the bean and delegate simply reuse the confluence management component, and as such, they are also effectively thread-safe.

Overall, the lack of thread-safety comes from the non-thread-safe entities. To handle scenarios where they would not be used in a thread-safe manner would require the components that own them to make them thread-safe.

The EJB will have CMT with each method using the Required level.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5+, J2EE 1.5
- Compile target: Java 1.5, Java 1.6, J2EE 1.5
- Application server: JBoss 4.2.2+

2.2 TopCoder Software Components

- Confluence Management 1.0
  - Provides the interface extended and used by this component, as well as the data objects.
- Logging Wrapper 2.0
  - Used for logging operations. Used in the ConfluenceManagementServiceBean and ConfluenceManagerWebServiceDelegate.
- Configuration API 1.0

    - The substitution to the ConfigManager. It provides the configuration parameters for all classes. Used in the ConfluenceManagementServiceBean and ConfluenceManagerWebServiceDelegate.
- Configuration Persistence 1.0.1
    - Provides file-based configuration that provides configuration as a ConfigurationObject. Used in the ConfluenceManagementServiceBean and ConfluenceManagerWebServiceDelegate.
- Object Factory 2.1.2 and Object Factory Configuration API Plugin 1.0
    - The component that will perform object creation in the factories. The plugin will simply provide a Configuration API-based SpecificationFactory for the ObjectFactory. Used in the ConfluenceManagementServiceBean and ConfluenceManagerWebServiceDelegate.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Names

com.topcoder.confluence.webservice
com.topcoder.confluence.webservice.bean
com.topcoder.confluence.webservice.client
com.topcoder.confluence.webservice.delegate

3.2 Configuration Parameters

3.2.1 ConfluenceManagementServiceBean configuration in initialize method

`Child ConfigurationObjects in root`

Child name	value	Required
spec_factory_config	The configuration to be put into the configuration object specification factory.	No

`Properties in root`

property name	value	required
log_name	The name of the log	No
confluence_manager_key	The name of the key to use with the object factory to instantiate a ConfluenceManager	Yes

3.2.2 ConfluenceManagerDelegate configuration constructor

`Child ConfigurationObjects in root`

Child name	value	Required
spec_factory_config	The configuration to be put into the configuration object specification factory.	No

`Properties in root`

property name	value	required

log_name	The name of the log	No
service_client_key	The name of the key to use with the object factory to instantiate a ConfluenceManagementServiceClient	Yes

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

4.2.1 Deploying the web service

- Extract the component distribution
- Follow Dependencies Configuration.
- Build the EAR file containing the web service
- Copy the EAR file to the JBOSS_INSTALL_PATH/server/default/deploy directory
- Start JBoss

4.2.2 Generating the WSDL and web services artifacts

The WSDL description and all other artifacts for the web service will automatically be generated by JBoss at deploy time. The URL to the WSDL file may be found on the following page provided by JBoss, which lists all the deployed web services:

```
http://[SERVER_HOST]:[SERVER_PORT]/jbossws/services
```

4.3 Demo

None

4.3.1 Setup

The setup for the EJB follows the usual steps. Follow the steps in 4.2 to set up the service. Pick an end point address and apply it to the service client configuration (the `@WebServiceClient` annotation).

The injection parameters should be specified in the EJB deployment descriptor (ejb-jar.xml) for the ConfluenceManagementServiceBean session bean. Below is an example of how this may be done.

```
<enterprise-beans>
<session>
       <ejb-name>ConfluenceManagementServiceBean</ejb-name>
       <ejb-class>
com.topcoder.confluence.webservice.bean.ConfluenceManagementServiceBean
       </ejb-class>
       <env-entry>
              <env-entry-name>confluenceManagerFile</env-entry-name>
              <env-entry-type>java.lang.String</env-entry-type>
              <env-entry-value>
                     bean_config.properties
```

```
				</env-entry-value>
			</env-entry>
			<env-entry>
			<env-entry-name>
				confluenceManagerNamespace
			</env-entry-name>
			<env-entry-type>java.lang.String</env-entry-type>
			<env-entry-value>
				ConfluenceManagementServiceBean
			</env-entry-value>
		</env-entry>
	</session>
	</enterprise-beans>
```

The configuration persistence will require two simple parameters – the name of the log, and the relevant object factory key – and the object factory configuration itself. It is a straightforward task of taking the information in section 3.2 and creating the configuration files for this component.

For the demo we will currently assume the following location of the WSDL file. The developer is free to alter this once the JBoss deployment takes place:

```
String address = “http://127.0.0.1:8080/confluence_services-
confluence_services-ejb/ConfluenceManagementServiceBean?wsdl”);
```

4.3.2 *Delegate demo*

This demo is a basic demonstration of the methods in Confluence Management via the delegate. The point of this is not to show a demo of manager usage, which is amply demonstrated in the management component, but to show that the delegate usage is the same, which is the point of the delegate.

```
        // create a ConfluenceManagerWebServiceDelegate using the
constructor with filename and namespace
        ConfluenceManagerWebServiceDelegate confluenceManager =
            new
ConfluenceManagerWebServiceDelegate(ConfluenceManagerWebServiceDelegate
.DEFAULT_CONFIG_PATH,
                "demo");

        // log in to Confluence and retrieve the needed token to
perform the operations
        String token = confluenceManager.login("user", "password");
        // if the user or password are not valid in Confluence, an
appropriate exception will be thrown

        // create a new page in confluence with CompontType - CUSTOM
        ConfluencePageCreationResult confluencePageCreationResult =
            confluenceManager.createPage(token, "Erebus Token Web
Services", "1.0",
                ConfluenceAssetType.COMPONENT_DESIGN,
ConfluenceCatalog.DOT_NET, ComponentType.CUSTOM);

        // create a new page in confluence with application code
        confluencePageCreationResult =
            confluenceManager.createPage(token, "Erebus Payment Web
Services", "1.0",
                ConfluenceAssetType.COMPONENT_DESIGN,
ConfluenceCatalog.JAVA, "12345");

        // create another page in confluence with page
        confluencePageCreationResult =
confluenceManager.createPage(token, new Page());

        // retrieve a page with ComponentType
        Page page =
            confluenceManager.retrievePage(token, "Erebus Token Web
Services", "1.0",
                ConfluenceAssetType.COMPONENT_DESIGN,
ConfluenceCatalog.DOT_NET, ComponentType.CUSTOM);

        // retrieve a page with applicationCode
        page =
            confluenceManager.retrievePage(token, "Erebus Token Web
Services", "1.0",
                ConfluenceAssetType.COMPONENT_DESIGN,
ConfluenceCatalog.JAVA, "12345");

        // log out from the Confluence:
        confluenceManager.logout(token);
        // if the token is not valid in Confluence, an appropriate
exception will
              // be thrown
```

4.3.3 Service client demo

This demo will briefly show how to use the ConfluenceManagementServiceClient directly. We will assume that

```
// create a ConfluenceManagementServiceClient using the default qualified
name and a specific address of the service, and obtain a proxy:
URL url = new URL(address);
ConfluenceManagementServiceClient client = new
ConfluenceManagementServiceClient(url);
ConfluenceManagementService service =
client.getConfluenceManagementServicePort();
// we can now access the service as we did in 4.3.3
```

5. Future Enhancements

If the Confluence Management component is updated, this component will be updated to reflect the changes.