Confluence Management Requirements Specification

1. Scope

1.1 Overview

This component provides a framework for adding pages to Confluence, as well as retrieving information about pages in Confluence. Access to the Confluence service will be through the Confluence SOAP API, and authorization / authentication will happen through that API as well.

Information about the Confluence SOAP API can be found here:

http://confluence.atlassian.com/display/DOC/Remote+API+Specification

http://confluence.atlassian.com/display/CONFEXT/Confluence+Command+Line+Interface

http://confluence.atlassian.com/rpc/soap-axis/confluenceservice-v1?wsdl

1.2 Logic Requirements

This component must implement all the functionality shown on the "Manager Class Diagram" in the provided interface TCUML.

1.2.1 Interfaces

The following interface will be provided by this component:

- ConfluenceManager

o This interface defines functionality for logging a user in and out of Confluence, as well as creating new documentation pages for TC components and projects, as well as retrieving information about pages created.

1.2.2 Accessing the SOAP services

Sample code in accessing the SOAP services can be found through the Confluence links above. If the designer wishes, Axis can be used in this design to generate the SOAP client class used to access the Confluence SOAP services.

1.2.3 Authorization

The authorization for users to be able to retrieve information and create pages in specific locations will be handled by Confluence. Before making an API call, the caller is expected to login with a username and password, returning a token. That same token is then passed to the subsequent calls into Confluence. This token is used to verify the user has access to the specific areas being manipulated. If a user attempts an unauthorized call, a ConfluenceNotAuthorizedException should be thrown.

1.2.4 Mapping

The DefaultConfluenceManager class contains a mapping of ConfluenceAssetType to base page locations. These locations are used when creating the page, as part of the page path creation. These are configurable, but if no custom values are given, the following values should be used as defaults:

- Component Design http://www.topcoder.com/wiki/display/docs/Design
- Component Development http://www.topcoder.com/wiki/display/docs/Development
- Application Specification http://www.topcoder.com/wiki/display/projects/$CODENAME$
- Application Architecture http://www.topcoder.com/wiki/display/docs/Architecture
- Application Assembly http://www.topcoder.com/wiki/display/docs/Assembly
- Application Testing http://www.topcoder.com/wiki/display/docs/Testing

For application specification, the code name value should be taken from the "applicationCode" string provided. If no string is provided, an exception should be thrown.

The Base Page is located in the wiki based on the space mapping described above. The page name is constructed per the table below:

Type	Page Name
Component Base Page	Catalog+Type+Name
Component Version Page	Catalog+Type+Name+Version
Application Base Page	ApplicationCode+Name
Application Version Page	ApplicationCode+Name+Version

1.2.5 Creating the page

The flow for creating a page can have a couple of different paths. When creating a page, the user provides the base component name and a version. If the base component already exists, a new version page is added (with its contents filled from a template), but if the base component page doesn't already exist, it is first created (with its contents filled from a template), and then the version page is added under it.

1.2.6 Templates

When a page is created, its contents should be initially filled with the information from a template page. The DefaultConfluenceManager class contains a mapping of ConfluencePageType to string locations. The template page locations are just basic pages in Confluence with default contents. To apply the template, the template page should be loaded from Confluence, its contents should be retrieved, and then the retrieved contents should be copied to the page being created.

1.2.7 Connection

The connection to Confluence could conceivably be lost at various points during the usage of this component. If this happens, a ConfluenceConnectionException should be thrown, and on each subsequent calls, an attempt should be made to reestablish the connection. If the reestablishment of the connection fails, another ConfluenceConnectionException should be thrown. This way, when the connection is available again, the API will function as normal.

1.2.8 Logging

Each manager method called should be logged at the DEBUG level, including the parameter information. If errors occur in any call, a message should be logged at the WARNING level before the exception is re-thrown.

1.2.9 Configuration

Configuration must occur through the Configuration API and the Configuration API Object Factory Plugin. Besides the normal ConfigurationObject constructors for the classes that involve configuration, constructors should be added that use the Configuration Persistence component to retrieve configuration information saved in configuration files.

1.2.10 Entities

All entities in the com.topcoder.confluence.entities package, and the two result classes, in the interface diagram should be provided. They are just basic data holders that represent the entities used by the framework. They should all implement the Serializable interface. Also, each entity must be serializable to XML, for use with the Confluence Service component. This should be done through the use of class annotations, like

```
"@XmlAccessorType(XmlAccessType.FIELD)", "@XmlType(name = "..." propOrder ={ "..." })"
```

etc...

1.3 Required Algorithms

The flow of creating the page, with the various different conditions, must be shown and described in the CS.

1.4 Example of the Software Usage

This component will be used as a direct API for manipulating confluence information.

1.5 Future Component Direction

Further manager implementations can be added.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

Designs must adhere to the interface diagram definition found in the architecture TCUML file provided [Confluence_Service_Architecture.tcuml]. Designers can choose to add more methods to the interfaces, but must keep the ones defined on the diagram as a minimum. Changes to the interfaces should be approved in the forum.

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.topcoder.confluence

com.topcoder.confluence.entitites

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The log to use
- The connection information for the Confluence API
- The mapping of template and location strings for the different page types

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

Confluence 2.7

SOAP

3.2.2 TopCoder Software Component Dependencies:

- Configuration API 1.0
- Configuration Persistence 1.0.1
- Base Exception 2.0
- Logging Wrapper 2.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

Confluence 2.7

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003
- Informix 10.0

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of the TopCoder UML Tool.