JIRA Management 1.0 Component Specification

1.Design

This component provides a framework for adding projects and versions to JIRA, as well as retrieving information about pages in JIRA. Access to the JIRA service will be through the JIRA SOAP API, and authorization / authentication will happen through that API as well.
Information about the JIRA SOAP API can be found here:
http://confluence.atlassian.com/display/JIRA/Creating+a+SOAP+Client
http://docs.atlassian.com/software/jira/docs/api/rpc-jira-plugin/latest/com/atlassian/jira/rpc/soap/JiraSoapService.html

1.1Design Patterns

DAO Pattern – this component provided a manager interfaces and implementations, which follows this pattern.

Strategy pattern – the manager class created in this design use the JiraSoapService in a strategic manner.

DTO pattern – all the entities used in this design are Data Transfer Objects.

Wrapper pattern – the **manager class** delegates their operations to the configured JiraSoapService.

1.2Industry Standards

- SOAP

1.3Required Algorithms

The sections below show the interactions needed by the JIRA Manager from this design.

1.3.1Entities and Hibernate

The two result classes (JiraProjectRetrievalResult, and JiraProjectCreationResult), as well as the entity classes (JiraVersion, JiraProject) are just basic data holders that represent the entities used by the framework.
They all implement the Serializable interface. Also, each entity is serializable to XML, for use with the JIRA Service component. This should is done through the use of class annotations, like:
`@XmlAccessorType(XmlAccessType.FIELD),`

`@XmlType(name = "..." propOrder = { "..." })`

1.3.2Logging

1.3.2.1The Log field can be initialized using:

If a logName is provided (not null or empty String)

```
log = LogManager.getLog( logName )
```

or:

`log = LogManager.getLog()` if no logName is provided

Logging is used in this way:

```
logger.log( Level.DEBUG, “Some internal details” );
```

1.3.2.2Logging can be turned on or off using:

`setLog(newLog)`method. If `newLog` is null, logging is disabled. To enable logging, a valid value should be provided.

1.3.2.3Determine the type of the logging:

```
//Determine if logging should be performed:
if( !log ==null)
       // determine the type of logging:
       if (verboseLogging== true)
       //detailed logging actions should be performed
       else
       //standard logging should be performed
else.
//if false: no logging should be performed.
```

1.3.2.4There are two types of logging that can be set using:

`setVerboseLogging(verboseLogging:boolean)` method:

- ***Standard log*** **that logs only the following:**

  `if(getVerboseLogging()==false):`
  - Method entry and parameter information (`Level.DEBUG` level);
  - Any exception `(Level.WARN level)` before the exception is re thrown. The exception name and stack trace should be logged.
    - ***Detailed log*** that logs exceptions and other details:

  `if(getVerboseLogging()==true):`
  - Entering the method and timestamp (`Level.DEBUG level`);
  - Method arguments (`Level.DEBUG level`);
  - Time spent in the method (`Level.DEBUG level`);
  - Return value (`Level.DEBUG level`).
  - Any exception `(Level.WARN level)` before the exception is re thrown. The exception name and stack trace should be logged.

1.3.3Schemes Mappings

The DefaultJiraManager class contains mappings of ContestType to string names, used to retrieve the notification scheme, permission scheme, and issue security scheme values for each project created.
When the project is created, the notification scheme, permission scheme, and issue security scheme are required to be set in the RemoteProject instance used.
If no mapping is given for a specific ContestType for any of the scheme names, the following defaults should be used:

Notification Scheme Names table (***notificationSchemes*** map):

Key	Value
`Generic`	`"Component Management Notification Scheme"`
`Custom`	`"Component Management Notification Scheme"`
`Application`	`"Client Project Notification Scheme"`

Permission Scheme Names table (***permissionSchemes*** map):

Key	Value
`Generic`	`"Generic Component Permission Scheme"`
`Custom`	`"Custom Component Permissions Scheme"`
`Application`	`"Client Projects"`

Issue Security Scheme Names table (***issueSecuritySchemes*** map):

Key	Value
`Generic`	`"Generic Component Issues Scheme"`
`Custom`	`"Custom Component Issues Scheme"`

Application	"Client Project Issues Scheme"

Please note that the following convention is used in the tables from above (key column):

Keys/Values table

Key	Real value
Generic	ComponentType.**GENERIC_COMPONENT**
Custom	ComponentType.**CUSTOM_COMPONENT**
Application	ComponentType.**APPLICATION**

1.3.4JIRA Management Operations

This section describes the all the available JIRA Management operations provided by default implementation of JiraManagement (DefaultJiraManagement):

1.3.4.1+login(username:String,password:String):String

```
// Call the namesake method from the configured JiraSoapService:
return jiraSoapService.login(userName, password);
```

1.3.4.2+logout(token:String):void

```
// Call the namesake method from the configured JiraSoapService:
jiraSoapService.logout(token);
```

1.3.4.3+createProject(token:String,project:JiraProject,version:JiraVersion,type: ComponentType):JiraProjectCreationResult

Note: Also please see "JIRA Management - Create A New JIRA Project - Sequence Diagram" for a better understanding of those interactions

```
// Convert the version to the JIRA entity:
RemoteVersion remoteVersion = convertToRemoteVersion(version);

// Convert the project to the JIRA entity:
RemoteProject remoteProject = convertToRemoteProject(project);

// Call the appropriate method from JiraSoapService to retrieve
// the RemoteProject:
RemoteProject tempRemoteProject =
jiraSoapService.getProjectByKey(token, projectKey);

// START IF the project does not exists (tempRemoteProject is null):
IF
        // Get the notification scheme name:
        String notificationSchemeName = notificationSchemes.get(type);
        // Get the permission scheme name:
        String permissionSchemeName = permissionSchemes.get(type);
        // Get the issue security scheme name:
        String issueSecuritySchemeName = issueSecuritySchemes.get(type);

        // Get the notification schemes:
        RemoteScheme[] notificationSchemes =
        jiraSoapService.getNotificationSchemes(token);

        // Get the permission schemes:
        RemotePermissionScheme[] permissionSchemes =
        jiraSoapService.getPermissionSchemes(token);

        // Get the issue security schemes
        // (only Enterprise edition supports security schemes)
```

```
        RemoteScheme[] issueSecuritySchemes =
        jiraSoapService.getSecuritySchemes(token);

        // Get the notification, permission, and issue security schemes
        // that match the names retrieved:

        // Get the notification scheme that matches the notification scheme
        //  name:
        RemoteScheme notificationScheme;
        FOR EACH tempNotificationScheme in notificationSchemes
              if(tempNotificationScheme.getName().equals( notificationSchemeNam
              e)) {
                    notificationScheme = tempNotificationScheme;
                    break;
              }
        END FOR EACH;

        // Get the permission scheme that matches the permission scheme name:
        RemotePermissionScheme permissionScheme;
        FOR EACH tempPermissionScheme in permissionSchemes
              if(tempPermissionScheme.getName().equals(  permissionSchemeName))
              {
                    permissionScheme = tempPermissionScheme;
                    break;
              }
        END FOR EACH;

        // Get the issue security scheme that matches the issue security scheme
        // name:
        RemoteScheme issueSecurityScheme;
        FOR EACH tempIssueSecurityScheme in issueSecuritySchemes
              if(tempIssueSecurityScheme.getName().equals( issueSecuritySchemeN
              ame)) {
                    issueSecurityScheme = tempIssueSecurityScheme;
                    break;
              }
        END FOR EACH;

        // Create a new project to save:
        remoteProject = new RemoteProject();
        // Set the obtained schemes:
        remoteProject.setNotificationScheme(notificationScheme);
        remoteProject.setPermissionScheme(permissionScheme);
        remoteProject.setIssueSecurityScheme(issueSecurityScheme);

END IF

// using remoteProject's setters, set all its properties with the
// corresponding values from the project argument retrieved using appropriate
// getters (see 1.3.5 section JiraProject to/from RemoteProject mapping table
// for all the available properties and mapping)

// create the project (or update if project already exists):
remoteProject = jiraSoapService.createProjectFromObject(token, remoteProject);
remoteProject = jiraSoapService.updateProject(token, remoteProject);

// Check if the version already exists:
RemoteVersion[] remoteVersions =
jiraSoapService.getVersions(token, project.getKey());
RemoteVersion remoteVersion;
boolean exists = false;
FOR EACH tempRemoteVersion in remoteVersions
      if(tempRemoteVersion.getName().equalsIgnoreCase(version.getName())) {
            exists = true;
            break;
      }
```

```
END FOR EACH;

// using remoteVersion's setters, set all its properties with the
// corresponding values from the version argument retrieved using appropriate
// getters (see 1.3.5 section JiraVersion to/from RemoteVersion mapping table
// for all the available properties and mapping)

// Add the version whether or not the version already exists:
remoteVersion = jiraSoapService.addVersion(token, project.getKey(),
remoteVersion);

// Convert the added RemoteVersion to a local version:
JiraVersion jiraVersion = convertToLocalVersion(remoteVersion);

// Convert the created RemoteProject to local JiraProject:
JiraProject jiraProject = convertToLocalProject(remoteProject);

// Create a new JiraProjectCreationResult and return the obtained
// JiraProjectCreationResult:
// if the project and version were found
return new JiraProjectCreationResult (jiraProject, jiraVersion,
JiraProjectCreationAction.PROJECT_AND_VERSION_EXISTED);

// if the project was found but the version was not found
// (version was created in this case):
return new JiraProjectCreationResult (jiraProject, jiraVersion,
JiraProjectCreationAction.PROJECT_EXISTED_VERSION_CREATED);

// if the project was not found:
return new JiraProjectCreationResult (jiraProject, jiraVersion,
JiraProjectCreationAction.PROJECT_AND_VERSION_CREATED);
```

1.3.4.4+addVersionToProject(token:String,project:JiraProject,version:JiraVersion): JiraProjectCreationResult

```
// get project (if project doesn't exist then throw JiraManagerException):
RemoteProject remoteProject =
jiraSoapService.getProjectByKey(project.getKey());

// Convert the given RemoteProject to local JiraProject:
JiraProject jiraProject = convertToLocalProject(remoteProject);

// Call the appropriate method from the configured JiraSoapService
// to retrieve the RemoteVersions and then retrieve the remoteVersion
// with the given name:
RemoteVersion[] remoteVersions =
jiraSoapService.getVersions(token, project.getKey());
RemoteVersion remoteVersion;
boolean exists = false;
FOR EACH tempRemoteVersion in remoteVersions
      if(tempRemoteVersion.getName().equalsIgnoreCase(version.getName())) {
            exists = true;
            break;
      }
END FOR EACH;

// Convert the given version to JIRA entity:
remoteVersion = convertToRemoteVersion(version);

// Whether or not the version already exists:
remoteVersion = jiraSoapService.addVersion(token, project.getKey(),
remoteVersion);

// Convert the added version to a local version:
JiraVersion jiraVersion = convertToLocalVersion(remoteVersion);
```

```
// Create a new JiraProjectCreationResult and return the obtained
// JiraProjectCreationResult:
// if the project and version were found
return new JiraProjectCreationResult (jiraProject, jiraVersion,
JiraProjectCreationAction.PROJECT_AND_VERSION_EXISTED);

// if the project was found but the version was not found
// (jiraVersion was created in this case):
return new JiraProjectCreationResult (jiraProject, jiraVersion,
JiraProjectCreationAction.PROJECT_EXISTED_VERSION_CREATED);
```

1.3.4.5+getProject(token:String,projectKey:String):JiraProjectRetrievalResult

```
// Call the appropriate method from JiraSoapService to retrieve
// the RemoteProject:
RemoteProject remoteProject =
jiraSoapService.getProjectByKey(token, projectKey);

// if RemoteProject is not found (remoteProject is null), go to 3.
// Convert the RemoteProject to local JiraProject:
JiraProject jiraProject = convertToLocalProject(remoteProject);

// Create a new JiraProjectRetrievalResult and return the obtained
// JiraProjectRetrievalResult:
// if the project was found (jiraVersion is null in this case):
return new JiraProjectRetrievalResult(jiraProject, null,
JiraProjectRetrievalResultAction.PROJECT_FOUND_NOT_VERSION);

// if the project was not found
// (jiraVersion and jiraProject are null in this case):
return new JiraProjectRetrievalResult(null, null,
JiraProjectRetrievalResultAction.PROJECT_NOT_FOUND);
```

1.3.4.6+getProject(token:String,projectKey:String,versionName:String): JiraProjectRetrievalResult

```
// Call the appropriate method from JiraSoapService to retrieve
// the RemoteProject:
RemoteProject remoteProject =
jiraSoapService.getProjectByKey(token, projectKey);

// if RemoteProject is not found (remoteProject is null), go to 5.
// Convert the RemoteProject to local JiraProject:
JiraProject jiraProject = convertToLocalProject(remoteProject);

// Call the appropriate method from the configured JiraSoapService
// to retrieve the RemoteVersions and then retrieve the remoteVersion
// with the given name:
RemoteVersion[] remoteVersions =
jiraSoapService.getVersions(token, projectKey);
RemoteVersion remoteVersion;
FOR EACH tempRemoteVersion in remoteVersions
      if(tempRemoteVersion.getName().equals(versionName)) {
            remoteVersion = tempRemoteVersion;
            break;
      }
END FOR EACH;

// if RemoteVersion is not found (remoteVersion is null), go to 5.
// Convert the RemoteVersion to local JiraVersion:
JiraVersion jiraVersion = convertToLocalVersion(remoteVersion);

// Create a new JiraProjectRetrievalResult and return the obtained
// JiraProjectRetrievalResult:
// if the project and version were found
return new JiraProjectRetrievalResult(jiraProject, jiraVersion,
```

```
JiraProjectRetrievalResultAction. PROJECT_AND_VERSION_FOUND);

// if the project was found but the version was not found
// (jiraVersion is null in this case):
return new JiraProjectRetrievalResult(jiraProject, null,
JiraProjectRetrievalResultAction.PROJECT_FOUND_NOT_VERSION);

// if the project was not found
// (jiraVersion and jiraProject are null in this case):
return new JiraProjectRetrievalResult(null, null,
JiraProjectRetrievalResultAction.PROJECT_NOT_FOUND);
```

1.3.4.7+getProjectVersions(token:String,projectKey:String):List<JiraVersion>

```
// Call the appropriate method from the configured JiraSoapService
// to retrieve the RemoteVersions:
RemoteVersion[] remoteVersions =
jiraSoapService.getVersions(token, projectKey);

// Convert the RemoteVersions to local JiraVersions:
// Create a new list with JiraVersions:
List<JiraVersion> jiraVersions = new ArrayList<JiraVersion>();
JiraVersion jiraVersion;
FOR EACH remoteVersion in remoteVersions
      // Convert the remoteVersion to local JiraVersion:
      jiraVersion = convertToLocalVersion(remoteVersion);
      jiraVersions.add(jiraVersion);
END FOR EACH;

// Return the obtained jiraVersions list:
return jiraVersions;
```

1.3.4.8+projectExists(token:String,projectName:String):boolean

```
// Retrieve all the projects using the configured JiraSoapService:
RemoteProject[] remoteProjects =
jiraSoapService.getProjectsNoSchemes(token);
boolean exists = false;
FOR EACH remoteProject in remoteProjects
      if(remoteProject.getName().equals.ignoreCase(projectName)) {
            exists = true;
            break;
      }
END FOR EACH
return exists;
```

1.3.4.9+getProject(token:String,projected:Long):JiraProjectRetrievalResult

```
// Call the appropriate method from JiraSoapService to retrieve
// the RemoteProject:
RemoteProject remoteProject =
jiraSoapService.getProjectById(token, projectKey);

// if RemoteProject is not found (remoteProject is null), go to 3.
// Convert the RemoteProject to local JiraProject:
JiraProject jiraProject = convertToLocalProject(remoteProject);

// Create a new JiraProjectRetrievalResult and return the obtained
// JiraProjectRetrievalResult:
// if the project was found (jiraVersion is null in this case):
return new JiraProjectRetrievalResult(jiraProject, null,
JiraProjectRetrievalResultAction.PROJECT_FOUND_NOT_VERSION);

// if the project was not found
// (jiraVersion and jiraProject are null in this case):
return new JiraProjectRetrievalResult(null, null,
```

```
JiraProjectRetrievalResultAction.PROJECT_NOT_FOUND);
```

1.3.4.10+deleteProject(token:String,projectKey:String):void

```
// Call the namesake method from the configured JiraSoapService:
jiraSoapService.deleteProject(token, projectKey);
```

1.3.4.11+updateProject(token:String,project:JiraProject):JiraProject

```
// Retrieve existing project by key (without schemes)
RemoteProject curProject =
      jiraSoapService.getProjectByKey(token, project.getKey());
// Retrive existing project by id (with schemes)
curProject = jiraSoapService.getProjectByKey(token, curProject.getId());

// Convert the given project to JIRA entity:
RemoteProject remoteProject = convertToRemoteProject (project);

// use existing schemes:
remoteProject.setPermissionScheme(curProject.getPermissionScheme());
remoteProject.setNotificationScheme(curProject.getNotificationScheme());
remoteProject.setIssueSecurityScheme(curProject.getIssueSecurityScheme());

// Call the namesake method from the configured JiraSoapService:
remoteProject = jiraSoapService.updateProject(token, remoteProject);

// Convert the RemoteProject to local JiraProject:
JiraProject jiraProject = convertToLocalProject(remoteProject);

// Return the updated project:
return jiraProject;
```

1.3.4.12+releaseVersion(token:String,projectKey:String,version:JiraVersion):void

```
// Convert the version to the JIRA entity:
RemoteVersion remoteVersion = convertToRemoteVersion(version);

// Call the namesake method from the configured JiraSoapService:
jiraSoapService.releaseVersion(token, projectKey, remoteVersion);
```

1.3.4.13+archiveVersion(token:String,projectKey:String,versionName:String,archive:boolean): void

```
// Call the namesake method from the configured JiraSoapService:
jiraSoapService.archiveVersion(token, projectKey, versionName, archive);
```

1.3.5Local to/from Remote entities mapping

Local JiraProject and JiraVersion are simple mappings to corresponding remote entities from JIRA: RemoteProject and respectively RemoteVersion.
The mappings are shown in the following tables:

JiraProject to/from RemoteProject mapping table

JiraProject property	RemoteProject property	Java type
description	description	String
key	key	String
lead	lead	String
projectUrl	projectUrl	String
url	url	String
id	id	String
name	name	String

JiraVersion to/from RemoteVersion mapping table

JiraVersion property	RemoteVersion property	Java type
`releaseDate`	`releaseDate`	`Date/Calendar`
`sequence`	`sequence`	`long/Long`
`archived`	`archived`	`boolean`
`released`	`released`	`boolean`
`id`	`id`	`String`
`name`	`name`	`String`

1.3.6Connection

The connection to JIRA could conceivably be lost at various points during the usage of this component. If this happens, the manager should attempt **once** to reestablish the connection, and if that fails, a JiraConnectionException should be thrown, and on each subsequent call, an attempt should be made to reestablish the connection. If the reestablishment of the connection fails, another JiraConnectionException should be thrown, until the connection has been reestablished. This way, when the connection is available again, the API will function as normal.

1.3.7Authorization

The authorization for users to be able to retrieve information and create pages in specific locations will be handled by JIRA. 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 JIRA. This token is used to verify the user has access to the specific areas being manipulated. If a user attempts an unauthorized call, a JiraNotAuthorizedException should be thrown.

1.4Component Class Overview

1.4.1com.topcoder.jira

JiraManager
This interface represents the JiraManager interface.
This interface defines the methods available for the JiraManager interface: login, logout, create project, add version to project, get project using different criteria, get project versions, check if a project exists, delete project, update project, release version and archive version.

JiraProjectRetrievalResult
This class represents the JiraProjectRetrievalResult java bean. A JiraProjectRetrievalResult is a JavaBean that holds the properties resulted after JIRA Project retrieval and can contain a project, retrievalResult and version properties. This is a simple java bean with a constructor that is used to initialize the properties and a corresponding getter for each property.
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (implements Serializable interface).

JiraProjectCreationResult
This class represents the JiraProjectCreationResult java bean. A JiraProjectCreationResult is a JavaBean that holds the properties resulted after JIRA Project creation and can contain a project, actionTaken and version properties. This is a simple java bean with a constructor that is used to initialize the properties and a corresponding getter for each property.
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (implements Serializable interface).

JiraProjectRetrievalResultAction
This enum represents the JiraProjectRetrievalResultAction enum and is a holder for all of the available JIRA project retrieval result actions. This enum gives a detail about the result of JIRA Project retrieval.
This enum defines three constants containing the available JIRA project retrieval result actions:

PROJECT_AND_VERSION_FOUND, PROJECT_FOUND_NOT_VERSION and PROJECT_NOT_FOUND.

JiraProjectCreationAction
This enum represents the JiraProjectCreationAction enum and is a holder for all of the available JIRA project creation actions. This enum gives a detail about the result of JIRA Project creation. This enum defines three constants containing the available JIRA project creation actions: PROJECT_AND_VERSION_CREATED, PROJECT_EXISTED_VERSION_CREATED and PROJECT_AND_VERSION_EXISTED.

ComponentType

This enum represents the ComponentType enum and is a holder for all of the available component types. It is used to retrieve the notification, permission and security issue schemes. This enum defines three constants containing the available component types: GENERIC_COMPONENT, CUSTOM_COMPONENT and APPLICATION.

1.4.2com.topcoder.jira.managers

DefaultJiraManager
This class is DefaultJiraManager and is a default realization of JiraManager interface.
This class has a default no-arg constructor, constructors that use configurations and constructors that initialize all the fields using provided arguments.
It uses JIRA framework to provide the needed operations by delegating to namesake methods from the configured JiraSoapService to perform the operations.
This class implements the methods available for the JiraManager interface: login, logout, create project, add version to project, get project using different criteria, get project versions, check if a project exists, delete project, update project, release version and archive version.
LOGGING
All the available manager operations perform logging using the Logging Wrapper component.
AUTHENTICATION AND AUTHORIZATION
The authorization for users to be able to retrieve information and create pages in specific locations will be handled by JIRA. 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 JIRA. This token is used to verify the user has access to the specific areas being manipulated. If a user attempts an unauthorized call, a JiraNotAuthorizedException is thrown.

1.4.3com.topcoder.jira.managers.entities

JiraProject
This class represents the JiraProject java bean. It is a mapping of RemoteProject entity from JIRA. A JiraProject can contain a description, key, lead, projectUrl, url, id and name properties.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method). Also a constructor is provided to initialize some properties: +JiraProject(description:String,key:String,lead:String,projectUrl:String,url:String).
See CS 1.3.5 "JiraProject to/from RemoteProject mapping table" for local-remote entities properties mappings and available properties.
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (implements Serializable interface).

JiraVersion
This class represents the JiraVersion java bean. It is a mapping of RemoteVersion entity from JIRA. A JiraVersion can contain a releaseDate, sequence, archived, released, id and name properties.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method). Also a constructor is provided to initialize some properties: +JiraVersion(releaseDate:Date,sequence:long,archived:boolean,released:boolean).
See CS 1.3.5 "JiraVersion to/from RemoteVersion mapping table" for local-remote entities

properties mappings and available properties.
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (implements Serializable interface).

1.5Component Exception Definitions

1.5.1com.topcoder.jira

JiraManagerException
This exception is the base exception for all exceptions raised from operations performed on JIRA Project and Version entities.
This exception wraps exceptions raised from JIRA or from usage of the TopCoder components (excerpt exceptions from configurations).

JiraConnectionException
This exception signals an issue when an attempt to reestablish the connection to JIRA fails.

JiraNotAuthorizedException
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.

JiraProjectValidationException
This exception signals an issue if the needed JIRA project is invalid (it does not exists in JIRA).

JiraSecurityTokenExpiredException
This exception signals an issue if the token obtained for authorization expires or becomes stale.

1.5.2com.topcoder.jira.managers

JiraManagerConfigurationException
This runtime exception signals an issue if the configured value is invalid.
This exception wraps the underlying exceptions when accessing the configured values.

1.5.3java.lang.

IllegalArgumentException
This system exception is thrown if an argument is invalid (null or long ids <= 0), empty string etc.

1.6Thread Safety

This component is thread-safe due to the following reasons:

The **default implementation of JiraManager** is technically mutable but application will use this class in a a thread-safe manner and will avoid concurrency issues. Also the setters/getters will be used in a thread safety fashion by the application so they do not affect the thread safety of this class.

The **existing entities** will be used in a thread safety manner so that the thread safety should not be affected in a multi thread environment; the application can synchronize the calls to manager operations to ensure the thread safety of this component.

2.Environment Requirements

2.1Environment

- Sun Java 1.5 is required for development and Java 1.5 and Java 1.6 for compilation
- RedHat Linux 7.1
- Solaris 7
- Informix 10.0
- Windows 2000, 2003

2.2TopCoder Software Components

- **Logging Wrapper 2.0** – This component is used to perform the logging.
- **Configuration API 1.0:** This is used as the main abstraction for configuration.
- **Configuration Persistence 1.0.1:** This will be used as the actual specific configuration reader for this component.
- **Base Exception 2.0** – This component is used to as base exception for custom exceptions defined in this design.

2.3Third Party Components

- Axis 1.4
- JIRA 3.1.1

3.Installation and Configuration

3.1Package Name

- **com.topcoder.jira –** contains the JIRA Manager interface, the defined enums and result entities. Also contains the majority of the custom exceptions defined in this design;
- **com.topcoder.jira.managers -** contains the default implementation of the JIRA Manager interface and the configuration exception defined in this design;
- **com.topcoder.jira.managers.entities -** contains the local JIRA Project and Version entities defined in this design;

3.2Configuration Parameters

3.2.1Resources

Under the **jiraManagerNamespace**:

Parameter	Description	Values
log_name_token	Represents the configuration log name. ***Optional***.	**String**. Can be null or empty.
jira_url_token	Represents the jira url String. ***Optional***.	**String**. If exists must be valid URL.

Under the **notificationSchemesNamespace**:

Parameter	Description	Values
generic_token	Represents the configuration value for generic key. ***Required***.	**String**. Can not be null or empty.
custom_token	Represents the configuration value for custom key. ***Required***.	**String**. Can not be null or empty.
application_token	Represents the configuration value for application key. ***Required***.	**String**. Can not be null or empty.

Under the **permissionSchemesNamespace**:

Parameter	Description	Values
generic_token	Represents the configuration value for generic key. ***Required***.	**String**. Can not be null or empty.
custom_token	Represents the configuration value for custom key. ***Required***.	**String**. Can not be null or empty.
application_token	Represents the configuration value for application key. ***Required***.	**String**. Can not be null or empty.

Under the **issueSecuritySchemesNamespace**:

Parameter	Description	Values
generic_token	Represents the configuration value for generic key. ***Required***.	**String**. Can not be null or empty.
custom_token	Represents the configuration value for custom key. ***Required***.	**String**. Can not be null or empty.
application_token	Represents the configuration value for application key. ***Required***.	**String**. Can not be null or empty.

3.3Dependencies Configuration

- JIRA framework and TopCoder dependencies should be properly configured.

4.Usage Notes

4.1Required steps to test the component

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

4.2Required steps to use the component

- JIRA framework and TopCoder dependencies should be properly configured.

4.3Demo

The following demo shows the usage of the operations available over the provided JIRA Manager:

4.3.1Demo for DefaultJiraManager

```
// we have the following url:
String jiraUrl =
" http://localhost:8080/rpc/soap/jirasoapservice-v2";

// create a JiraManager using the default constructor:
JiraManager jiraManager = new DefaultJiraManager();
// we also can create a JiraManager using the provided url:
jiraManager = new DefaultJiraManager(jiraUrl, null, null, null, null);

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

JiraProjectRetrievalResult jiraProjectRetrievalResult;
JiraProject jiraProject, otherJiraProject...;
JiraVersion jiraVersion, otherJiraVersion, otherJiraVersion1 …;

// create a new Project in JIRA:
jiraManager.createProject(token, jiraProject, jiraVersion,
ComponentType.CUSTOM_COMPONENT)

// add version to project in JIRA:
jiraManager.addVersionToProject(token, jiraProject, otherJiraVersion);

// get project with the given key from JIRA:
jiraProjectRetrievalResult = jiraManager.getProject(token, "projectKey1");

// get project with the given key and version name from JIRA:
jiraProjectRetrievalResult = jiraManager.getProject(token, "projectKey2",
"1.01");

// get JIRA versions for JIRA project
```

```
List<JiraVersion> versions = jiraManager.getProjectVersions(token,
“projectKey2”);

// Check if a JIRA project exists:
boolean exists = jiraManager.projectExists(token, “projectKey2”);

// get project with the given id from JIRA:
jiraProjectRetrievalResult = jiraManager.getProject(token, new Long(111222));

// update a JIRA project:
jiraManager.updateProject(token, otherJiraProject);

// release a JIRA version:
jiraManager.releaseVersion(token, “projectKey2”, otherJiraVersion1);

// archive a JIRA version:
jiraManager.archiveVersion (token, “projectKey2”, “1.0”, true);


// delete a JIRA project:
jiraManager.deleteProject(token, “projectKey2”);


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

5.Future Enhancements

- Further manager implementations can be added.