Rally Connector for Atlassian Jira

Table of Contents

Connector for Atlassian Jira **3**

Connector for Jira Installation and User Guide **3**

Install the Connector for Jira 6

Jira Setup 7

Setup for Jira Connector 10

Configure the Jira Connector 11

XML Tags in the Jira Connector 16

Run the Connector for Jira 25

Jira Connector Workflows 26

Jira Connector Best Practices 32

Troubleshoot the Jira Connector 34

Jira Connector Revision History 37

Sample Configuration Files for Atlassian Jira **41**

Connector for Atlassian Jira

The Jira Rest connector provides bi-directional creation and updates of stories, defects, and tasks between and Jira. This provides an efficient means of using a separate defect tracking system without entering information twice.

NOTE

Be sure to review the capabilities of each of these options to ensure the proper use case for your needs.

- The Connector for Jira is better suited for defect management-specific use cases. Learn more.
- The Adapter for Jira supports more advanced features and use cases for teams practicing agile at scale. It is primarily for bi-directional syncing between stories and epics/portfolio items. Learn more.
- The OpsHub or Tasktop solution is recommended for bi-directional syncing of data between and Jira systems. Learn more.

Features:

- Runs these update and copy services:
  - Create defect from to Jira
  - Create issue from Jira to
  - Update defect from to Jira
  - Update issue from Jira to
  - Update fields and Jira fields
- Updates the name, description, state, priority, and owner fields
- Abides by the (default) Jira Workflow
- Supports proxy servers
- Creates a weblink between the defect and the Jira issue

See the Connector for Jira Installation and User Guide for information on installing, configuring, and using the connector.

Supported Versions

Version	End of Support
Jira 8	March 15, 2022
Jira 7	November 28, 2021

See Supported Versions for Connectors for more information.

Cost: Free

Connector for Jira Installation and User Guide

The connector for Jira allows customers to use for Agile lifecycle management while still using Jira for defect management.

This connector supports Jira 7 or greater (server) and Jira OnDemand (SaaS). The Jira connector provides the ability to reflect work items between Jira and . The connector runs on a computer behind your firewall using Ruby and the Jira REST API. Most fields between the two systems can be mapped. The connector does not support mapping Milestone information, portfolio items, or bi-directional creation or update of features to Jira epics at this time.

WARNING
It is important to recognize that the data models of and Jira are not identical and, as such, the two systems are only partially compatible. We recommend that you take some time to identify the key data items you want to track in both systems and consider what you want the policy to be as far as having a primary system of record. This guide discusses some of the trade-offs to be considered and some potential approaches.

The connector is configured through an XML file. A log file is created to track all changes made in and Jira by the connector. The connector requires that a custom field exists in each system to store the unique ID of the linked objects in the other system. The connector copies fields from or Jira based on a field mapping specified in the configuration file. Standard and custom fields can be mapped between the two systems.

The connector provides five services to map objects between and Jira:

- Copy issues created in Jira to work items
- Copy work items created in to Jira issues
- Update work items based on changes made to Jira issues
- Update Jira issues based on changes made to work items
- Update work items and Jira issues

The configuration file specifies which services to run, and in what order.

An alternative update service, UPDATE_RALLYFIELDS_AND_OTHER, first searches for updates and pushes only the changed fields to the Jira system. It then searches the Jira system for updated artifacts, and pushes all mapped fields of those artifacts into the system. This reduces the chances of data overwrites. The only scenario where an overwrite might occur is if a change is made in and a change is made in Jira (in reference to two artifacts previously associated together). In this scenario, first any mapped field in which has been modified is copied to Jira (potentially overwriting a previously modified mapped field in Jira), then all mapped fields are copied from Jira to .

This section includes the following topics:

- Software and Hardware Requirements
- Jira Connector Installation Prerequisites
- Install the Jira Connector
- Jira Setup
- Setup for the Jira Connector
- Configure the Jira Connector
- XML Tags
- Run the Jira Connector
- Jira Connector Workflows
- Jira Connector Best Practices
- Troubleshoot the Jira Connector
- Jira Connector Revision History

The connector supports mapping top-level Jira issues (bug, new feature, improvement, task, or a custom issue type) but not Jira sub-level types (for example, sub-task). We recommend mapping the defect work item type to Jira bug and user story work item type to the other Jira top-level issue types.

The remainder of this guide assumes mapping the defect work item type to a Jira bug issue type. If you are mapping another Jira issue type, you will need to have a modified `jira_config.xml` file to accommodate that scenario. In general, you can follow this document if your scenario differs from the one presented here. For example, you can substitute your Jira issue type for bug or user story for defect when referencing the work item type.

Before you begin, review the following items:

- The following people will be involved in getting fields set up in both and Jira:

  - A Jira administrator user who can create a custom field on Jira issue types.
  - A administrator user (subscription or workspace) who can create a custom field on the artifact you want to map with Jira.
- Consider the process you want to set up between and Jira:
  - What problem are you trying to solve?
  - Where do objects start and end their lifecycle?
- Which fields need updated? Identify a test project in Jira to use for testing.
- Identify a test workspace and project to use for testing.
- Decide on which computer the connector will run.

The Integrations Best Practices contains some useful information for connector setup.

Software and Hardware Requirements

The following are the hardware and software requirements to install and run the connector for Jira:

- A subscription
- Windows 10, Windows 2016 Server, or Linux/Mac operating systems
- A working instance of Jira Server (7.0 or greater) or a Jira OnDemand instance
- Ruby 2.2.6 installed
  - For Linux/Mac operating systems, download .gz for ruby 2.2.6 (we recommend installing using rvm or rbenv). Select the "Install Td/Tk support" "Add Ruby executables to your Path" and "Associate .rb and .rbw files with this Ruby installation" options when prompted.
  - For Windows operating system, download .exe for ruby 2.2.6 (x64). Select the "Install Td/Tk support" "Add Ruby executables to your Path" and "Associate .rb and .rbw files with this Ruby installation" options when prompted.
  - Add path to the ruby `bin` directory to the environment path variable.
- Ruby Development Kit for Ruby 2.0 – 2.3, only necessary for Windows
  - Download the Ruby DevKit: DevKit-mingw64-64-4.7.2-20130224-1432-sfx.exe
  - Add path to the devkit's bin directory to environment Path variable:
    a. `cd` to the Ruby DevKit root directory
    b. Run command: `ruby dk.rb init`
    c. Run command: `ruby dk.rb install`
- The Jira connector does not support LDAP.

Jira Connector Installation Prerequisites

Before you begin the installation, be sure you have the following:

- Access to a test environment installation of Jira
- administrator privileges (subscription or workspace) are needed for setup, but only user access is needed to run the connector.
- Proxy server details if the machine used to run the connector uses a proxy server to connect to or Jira
- A Jira user with administrator privileges

Set the HTTP_Proxy Variable

(Optional) If your site uses a proxy server to access , you will need to set the **http_proxy** environment variable. The following procedure is for a Windows platform.

1. From **Start**, right-click on **Computer**, then select **Properties.**
2. Select **Advanced system settings**.
3. On System Properties, select **Environment Variables.**

4. On Environment Variables, select **New** under System Variables.
5. On New System Variables, enter *http_proxy* in the Variable name: field, then enter your proxy's full HTTP URL in the **Variable value** field (for example, `http://www-cache:8000`).
6. Select **OK** (three times) to save and exit this process.
   You may need to restart your system in order for the change to take effect.

Install the Connector for Jira

Before installing the connector be sure you have completed all the prerequisites.

1. Download the connector by following the steps here.
2. Extract the contents of the zip file `CAAgileCentralConnectorforJira-x.y.z-master-bbb.zip` locally on your machine (such as `C:\rally` for Windows or `/Users/username/Downloads/` on a MAC).

   As a result of unpacking of the distribution archive, the following files and folders are created:

   - `field_handlers` — A folder for custom field handers
   - `jira_config.xml` — Sample configuration file
   - `jira_issue_workflow.xml` — Sample configuration file for Jira Bug workflow steps
   - `jira_workflow.xml` — Sample configuration file for Jira workflow steps
   - `install_gems.rb` — Ruby script to install the necessary gems for this connector
   - `ca_agile_central2_jira_connector.rb` — Executable to run the connector (located with the rallyeif-jira gem)
3. Change the directory to the root of the connector installation: `cd CAAgileCentralConnectorforJira-4.x-master-z`
4. Set environment variable GEM_COMMAND to gem executable located in Ruby installation directory. Here are location examples on Linux and Windows, respectively:

   - `/myhome/.rvm/rubies/ruby-2.2.6/bin/gem`
   - `C:\Ruby226\bin\gem`
5. Run the `install_gems.rb` Ruby script to get all associated gems installed.

   If you see output similar to *"You don't have write permissions ..."* then you will need to either:

   - Consult with your system administrator to obtain the necessary permissions (typically done by using sudo or other similar software).
   - Have the system administrator run the `install_gems.rb` command. To run the command in a console or terminal window at the prompt, type the following:
     ```
     ruby install_gems.rb
     ```

   Sample output:

   ```
   ruby install_gems.rb
       rubygems-update installed
       xml-simple installed
       httpclient installed
       multipart-post installed
       faraday installed
       mime-types installed
       rally_api installed
       rallyeif-wrk installed
       rally_hp_alm_api installed
       rallyeif-qc installed
   ```

Jira Setup

The Jira connector user (as specified in the configuration file) does not need to be part of the jira-admin group, but they must have permissions for the following operations:

- Add comments
- Assign issues
- Assignable user
- Browse projects
- Close issues
- Create attachments
- Create issues
- Edit issues
- Modify reporter
- Resolve issues
- Schedule issues

NOTE

- The Jira configuration can become quite complex. We have found that it is good to use the Jira Permission Helper button to check that all is correct. All eleven of the above permissions must show a green check mark in the Permission Helper before using the connector.
- As a secondary test, log in to your Jira instance as the connector user. Try to edit an existing issue. If this is not allowed, then most likely the permissions are not correct for the connector to run.

In most cases, a Jira user in the jira-developers group has the above permissions. If one of the above permissions is absent, the log file will likely show something like:

```
 INFO : JiraConnection.connect - Using RallyJest version 1.1.19
DEBUG : RallyJest::JiraComm.block in execute_request - issuing a GET request for endpoint:
                /rest/api/2/serverInfo
DEBUG : RallyJest::JiraComm.execute_request - {
                "baseUrl":"http://jira.jpkole.org:8080","version":"5.2.9","versionNumbers":[5,2,9],
                "buildNumber":852,"buildDate":"2013-03-12T00:00:00.000-0400",
                "serverTime":"2013-10-17T10:37:05.815-0400",
                "scmInfo":"31363faa193505dadf581a01e480744edf4ac0de",
                "serverTitle":"GMS Issue Tracker"}
 INFO : JiraConnection.connect - Connected to JIRA at http://jira.jpkole.org:8080, (version 5.2.9)
DEBUG : RallyJest::JiraComm.block in execute_request - issuing a GET request for endpoint:
                /rest/api/2//mypermissions?projectKey=JPKOLE
DEBUG : RallyJest::JiraComm.execute_request - {
                "permissions":{
                        "EDIT_ISSUE":{"id":"12","key":"EDIT_ISSUE",
                                "name":"Edit Issues","description":"Ability to edit issues.",
                                "havePermission":true},
                        "VIEW_VERSION_CONTROL":{"id":"29","key":"VIEW_VERSION_CONTROL",
                                "name":"View Issue Source Tab",
                                "description":"Allows users to view related source code commits ....",
                                "havePermission":true},
                        "MODIFY_REPORTER":{"id":"30","key":"MODIFY_REPORTER",
                                "name":"Modify Reporter",
                                "description":"Ability to modify the reporter when creating or ....",
                                "havePermission":false},
                        ....}}
```

```
ERROR : JiraConnection.initialize - JIRA Permissions incorrect for Modify Reporter
 INFO : JiraConnection.disconnect - Disconnected from JIRA
```

Use the following procedure to modify the permissions on a project (the procedure was developed on a Jira 6.2 SaaS instance).

1. Log in to Jira with an account which has appropriate privileges.
2. Select your desired project under the **Projects** tab.
3. Select **Administration** under the project name (instead of **Overview**).
4. Select **Permissions** in the left column.
5. Edit the permissions.

Determine How to Handle Closed Issues in Jira When Using Restrictive Flow

By default, Jira does not allow an issue (bug) with a status of Closed to be updated. When an issue is first copied from Jira to (the COPY_JIRA_TO_RALLY service), the connector needs to record the ObjectID of the newly created defect into the RallyID custom field of the originating Jira issue. However, if the issue is closed, the update will fail and the RallyID custom field in Jira will remain empty on that issue. The next time the connector runs, that issue will appear to be a new issue (not yet copied to), and the connector will copy it again, creating a duplicate. There are several ways to handle this:

- Ignore closed Jira issues:
  Adding a `<CopySelector>` tag in your configuration file within the `<JiraConnection>` section will prevent the connector from trying to copy closed Jira issues to . For example:

```
<Config>
        ....
        <JiraConnection>
                <CopySelectors>
                        <CopySelector>Status != Closed</CopySelector>
                        ....
                </CopySelectors>
          ....
```

- Change the configuration in Jira:
  Configure Jira to allow for the updating of closed issues. Follow the Atlassian instructions *Allow Editing of Closed Issues* to enable the modification of closed issues in Jira.
- By default, Jira 7 comes with simplified workflow that allows updating an issue (bug) in a final status to be updated. There is no need to use:

```
<CopySelector>Status != Done</CopySelector>
```

Create an External ID Field in Jira

In the Jira system, you must create a custom field which will be used to record the Object ID (OID) of the corresponding artifact. The underlying database for the Jira instance should support this External ID field up to 14 digits in length.

> **NOTE**
> You may need to contact your Jira administrator to perform the following steps.

The following procedure was created using a Jira 6.2-OD-03 instance.

1. On the Administration tab in the navigation panel, select **Custom Fields** in the Issue Fields section.
2. On the Custom Fields page, select **Add Custom Field** in the upper-right of the screen.
3. On the Select a Field Type page, select **Standard** in the left pane, then select **Number Field** in the right pane.
4. Select **Next**.
5. On the Configure Number Field page:
   a. Enter `RallyID` for the Name.

   b. Enter `Must default to -1` for the Description. (Not applicable if using Jira OnDemand.)
   c. Select **Create**.
6. On the Associate field RallyID to screens page, select ALL of the available screens. Typically, there are at least three (Default Screen, Resolve Issue Screen, and Workflow Screen), but there may be more.
7. Select **Update.**
   The Custom Fields page should display with your newly created custom field displayed in the list.
8. Under the gear icon to the right of your new custom field name, select **Configure**.
9. On the Configure Custom Field: RallyID page, select **Edit Configuration.**
10. On the Modify configuration scheme context page:
    a. In the Choose applicable issue types section, select **Any issue type** (or a specific, top-level issue such as sub-task, technical task, bug, epic, improvement, new feature, story, or task)
    b. In the Choose applicable context section, select Global context (or choose a specific project after selecting **Apply to issues under selected projects**).
    c. Select **Modify** at the bottom of screen.
11. On the Configure Custom Field: RallyID page, select **Edit Default Value**. (Not applicable if using Jira OnDemand.)
12. On the Set Custom Field Defaults page, enter **-1** in the box next to RallyID, then select **Set Default**. (Not applicable if using Jira OnDemand.)

Create a CrosslinkUrlField in Jira

It can be useful to have a selectable link in the Jira issue that will open the corresponding artifact in a browser window. The `<CrosslinkUrlField>` element in the `<JiraConnection>` section of the configuration file allows you to specify the name of the custom field that will contain this link (URL). Use the same process in Jira as used for the RallyID field:

- We suggest a field name of RallyLink or RallyURL
- Must be a URL Field or Free Text type of field
- This field name will be used in the `<CrosslinkUrlField>` element within the `<JiraConnection>` section of the configuration file
- This field will only populate during a copy, it will not populate during an update

Configure Jira Workflow to Allow Transitions to be Executed by Users Other Than Assignees

For example, default (read-only) Jira 7 workflow has a condition on the Start Progress, Stop Progress, and Resolve Issue transitions that restricts the ability to execute the transition to the issue's assignee. Since the connector user (the value of the `<User>` element in the `<JiraConnection>` section of the configuration file) will likely not be the assignee for many issues, the connector will fail to progress those issues through the workflow and update their status when copying or updating based on changes in . You may see the following error message if you encounter this issue:

```
ERROR : RallyEIF::WRK::Connector.rescue in copy_to_other - Unable to create a Jira Bug with these field
 values: {:Assignee=>"...", :Priority=>"High", :Reporter=>"...", :"Doc Enh Req"=>"No", :Summary=>"...",
 "RallyID"=>"99999999999", "RallyURL"=>"https://rally1.rallydev.com/slm/#/detail/defect/99999999999"},
 undefined method `has_key?' for nil:NilClass
```

If your usage scenario includes the default Jira workflow, COPY_RALLY_TO_JIRA, or UPDATE_RALLY_TO_JIRA, you will need to modify your workflow.If you are already using a custom Jira workflow, make it inactive, make edits as described to remove any conditions on transitions, and re-activate it.

1. Use the Copy link for the Jira workflow on the View Workflows page to create a copy of the default workflow.
2. On the Workflows page, select **Steps** in the new workflow to display the transitions.
3. Select the **Start Progress** transition. Under the Only assignee can execute this transition section, select **Delete**.
4. Repeat step 3 for the Stop Progress and Resolve Issue transitions.
5. Create a new workflow scheme to contain the new workflow and assign the new workflow to it for issue type Bug.
6. Associate the new workflow scheme with your test project.

Setup for Jira Connector

accesses fields from their Display Name, not their Name. Spaces and underscores are removed, for example, Example Field becomes ExampleField.

Create an External ID Field in

The External ID custom field in is used by the connector to record the key of the corresponding Jira issue.

1. Log into as a workspace or subscription administrator.
2. Select **Setup**

   from the menu bar in the upper-right corner of the display.
3. Select **Workspaces & Projects**.
4. Select the workspace that you wish to map to Jira. This will take you to the detail page for the given workspace.
5. In the left pane, select **Fields** and ensure the **Work Item Type** selected is **Defect** or **User Story** (whichever work item you are mapping).
6. Select **Actions, New Field**.
7. Enter a Name of JiraKey, Display Name of JiraKey (name and display name must match), and type of String. **Note:** You can choose a different name (like JiraID) for the custom field in , but the name you choose must conform to these rules:
   – Begins with an uppercase letter
   – Less than 41 characters (40 is the maximum for display name)
   – No underscores, dashes, or spaces
8. Select **Save & Close**.
9. Make note of the name of this field. Once you start using the connector, this will contain the Jira key (such as TST-213) of the Jira issue you are mapping between the two systems.

Create a CrosslinkUrlField in

It can be useful to have a selectable link in the artifact which will open the corresponding Jira issue in a browser window. The `<CrosslinkUrlField>` XML element in the `<RallyConnection>` section of the configuration file is where to specify the name of the custom field in which will contain this selectable link (URL). Use the same sort of process in as you used for the JiraKey field:

- The first character of the field name must be upper case.
- It is helpful to make the Name and Display Name identical.
- We suggest a field name of JiraLink.
- The field type must be String (not-selectable) or Text (selectable). A field type of Web Link does not function properly at this time.
- This field name will be used in the `<CrosslinkUrlField>` element within the `<RallyConnection>` section of the configuration file.
- This field will only populate when using the Copy service. The field will not populate during an update.

Set Up States and Resolutions

There are two additional setup steps required for the connector to successfully push issues through the Jira workflow. We recommend starting with a Jira project that uses the default Jira workflow. Once the connector is working successfully, you can transition to a custom workflow.

Create an Additional Reopened State for Work Items

1. Log into as a workspace or subscription administrator.
2. Select **Setup**

   from the menu bar in the upper-right corner of the display.
3. Select **Workspaces & Projects**.
4. Select the workspace that you wish to map to Jira. This will take you to the workspace's detail page.
5. In the left pane, select **Fields** and ensure the Work Item Type selected is **Defect** (or whichever work item you are mapping).
6. Find the State field and select its edit icon at the far right.
7. Add a State named **Reopened**.
8. Select **Save & Close**.

Make Choice Lists for Resolution Match Between the Two Systems

> **NOTE**
> By default, only defects have State and Resolution fields. If you want to use these fields with user stories, you must create custom fields for that work item type.

You can edit the list of resolutions in Jira from the Administration tab by selecting the Resolutions link in the Issue Setting section in the navigation panel on the left.Alternatively, you can use an OtherEnumFieldHandler to create a one-to-one mapping between resolutions and Jira resolutions. This approach is described more fully in Mapping Drop-Down Values.

1. Log into as a workspace or subscription administrator.
2. Select **Setup**

   from the menu bar in the upper-right corner of the display.
3. Select **Workspaces & Projects**.
4. Select the workspace that you wish to map to Jira. This will take you to the workspace's detail page.
5. In the left pane, select **Fields** and ensure the Work Item Type selected is **Defect** or **User Story** (whichever work item you are mapping).
6. Find the Resolution field and select its edit icon at the far right.
7. Edit the list of resolutions to make them match the corresponding list in Jira.
8. Select **Save & Close**.

Configure the Jira Connector

You'll need to create the configuration file for the Jira Connector using the syntax details.

If your instance of Jira does not allow issues to be created without an assignee, there are several ways to work around this behavior, including:

- Turn on Allow Unassigned Issues in Jira, under **System, General Configuration** on the Administration page.
- Set up a field handler for mapping users and explicitly map 's Owner to Jira's Assignee.
- Specify a default assignee by adding the following to your configuration file in the `<JiraConnection>` section as follows:

```
<Config>
        ....
        <JiraConnection>
                ....
                <FieldDefaults>
```

```
        <Field><Name>Assignee</Name><Default>someuser</Default></Field>
    </FieldDefaults>
    ....
```

Jira Configuration File Syntax

Each major section is surrounded by a tag to delineate that section:

RallyConnection Defines the connection information for , including URL, user, and password or APIKey.

JiraConnection Defines the connection information for Jira, including the Jira URL, user, password, and artifact type.

> **NOTE**
> As of the 4.7.x release of the JiraConnectorForAgileCentral, we have deprecated the use of the Rest modifier in `<JiraRestConnection>` and with the naming of the connector. Configuration files that currently use the `<JiraRestConnection>` tag will still work although the preferred tag is now `<JiraConnection>`. will support the use of either tag through the life cycle of 4.7.x but the recognition of the `<JiraRestConnection>` tag will not be supported upon the release of the 4.8.x series.

Connector Defines the field mapping between the two systems. Generally, strings should be mapped to strings, and integers to integers. In Jira, field names are generally specified with the first letter capitalized, such as Summary or Priority.

ConnectorRunner Specifies parameters related to the services the connector is to run and how often to run the services.

> **NOTE**
> Beginning with version 3.0.0 of the JiraConnector, all Jira fields mentioned in the configuration file (such as `jira_config.xml`) must be in terms of the Jira display name, identical to what is shown in the Jira GUI on the issue create, edit, and resolution panels.

Edit the Configuration File

A sample configuration file, `jira_config.xml` , is included in the `.exe` or `.zip` file and should be in the root directory of the installation. We recommend making a backup copy of the `jira_config.xml` in case you need to reference a valid configuration file later.

> **NOTE**
> Incrementally set up the connector! Start with a basic configuration file, ensure that you can connect to Jira and in a test environment. Once you validate this is set up correctly, then start customizing the field mapping and field handler sections.

Edit `jira_config.xml` and enter the appropriate values between each begin and end tag.

```
<Config>
        <RallyConnection>
                <Url>rally1.rallydev.com</Url>
                <WorkspaceName>Workspace Name</WorkspaceName>
                <Projects>
                        <Project>Project</Project> </Projects>
                <User>user@company.com</User>
                <Password>password</Password>
                <!-- You can use an APIKey in place of User and Password tags -->
                <!-- <APIKey>YourAPIKeyValueGoesHere...</APIKey>-->
                <ArtifactType>Defect</ArtifactType>
                <ExternalIDField>JiraKey</ExternalIDField>
        </RallyConnection>

        <JiraConnection>
```

```
                        <Url>http://jiraserver:port</Url>
                        <User>jirauser</User>
                        <Password>jirapassword</Password>
                        <ProxyURL>proxy_host:port</ProxyURL>
                        <ProxyUser>proxy_user_name</ProxyUser>
                        <ProxyPassword>proxy_password</ProxyPassword>
                        <ArtifactType>bug</ArtifactType>
                        <ExternalIDField>RallyID</ExternalIDField>
                        <Project>The Project Key (not the project name)</Project>
                        <WorkflowFile>jira_issue_workflow.xml</WorkflowFile>
                        <CopySelectors>
                                <CopySelector>Status != Closed</CopySelector>
                        </CopySelectors>
                </JiraConnection>

                <Connector>
                        <FieldMapping>
                                <Field><Rally>Name</Rally>          <Other>Summary</Other></Field>
                                <Field><Rally>Description</Rally>  <Other>Description</Other></Field>
                                <Field><Rally>State</Rally>           <Other>Status</Other></Field>
                                <Field><Rally>Resolution</Rally>    <Other>Resolution</Other></Field>
                        </FieldMapping>

                        <OtherFieldHandlers>
                                <JiraNewlineFieldHandler>
                                        <FieldName>Description</FieldName>
                                </JiraNewlineFieldHandler>

                                <OtherEnumFieldHandler>
                                        <FieldName>Status</FieldName>
                                        <Mappings>
                                                <Field><Rally>Submitted</Rally> <Other>Open</Other></Field>
                                                <Field><Rally>Open</Rally>        <Other>In Progress</Other></
Field>
                                                <Field><Rally>Reopened</Rally> <Other>Reopened</Other></Field>
                                                <Field><Rally>Fixed</Rally>        <Other>Resolved</Other></
Field>
                                                <Field><Rally>Closed</Rally>      <Other>Closed</Other></
Field>
                                        </Mappings>
                                </OtherEnumFieldHandler>
                        </OtherFieldHandlers>
                </Connector>

                <ConnectorRunner>
                        <Preview>false</Preview>
                        <Services>COPY_JIRA_TO_RALLY,  UPDATE_RALLY_TO_JIRA</Services>
                </ConnectorRunner>
        </Config>
```

Field Mapping Between <keyword keyref="product-name"></keyword> and Jira

The field mapping section is located between the `<Connector>` tags in the configuration file and defines which fields map between the two systems.

For example, the following setup defines a mapping between the fields Name, Description, and Priority, to the Headline, Description, and Priority fields in the other system (respectively). On a create or update, the connector will only work with fields specified in this `<FieldMapping>` section.

```
<Config>
     ....
     <Connector>
           <FieldMapping>
             <Field><Rally>Name</Rally>        <Other>Headline</Other></Field>
             <Field><Rally>Description</Rally> <Other>Description</Other></Field>
             <Field><Rally>Priority</Rally>    <Other>Priority</Other></Field>
           </FieldMapping>
           ....
     </Connector>
```

When you set up your mapping between the two systems, ensure the fields are compatible between the two systems (an integer field should map to an integer field in the other system, a rich text should map to a rich text in the other system).

Otherwise, you might experience situations where information is not created or updated between the two systems and you may see an error in the log file. For example, the connector will post an error for a particular work item if you try to send a string to a custom field of type integer in .

If you are mapping a drop-down value field, the connector assumes the drop-down values match. Otherwise, the connector will generate an error that the value was not found in the list. If your drop-down values are different between the two systems, see Map Drop-Down Values.

Other Multiple Value Fields

Jira has several fields that can have multiple values that are not as restrictive as State or Resolution fields. Affects Versions, Fix Versions, and Components fields in Jira can be bi-directionally mapped with fields, as long as enumfieldhandler is used to handle disparate values (see Field Directionality). In the course of mapping from Jira to , if there are multiple values for the field they will be copied to (or updated in) as a string of comma-separated values.

Field Handlers

- Map drop-down values
- Map user names
- Map reference fields from

CopySelector and UpdateSelector

In addition to standard copy and update selectors, the 4.7.0 and later versions of the Jira connector support selectors that allow subset and range operators using `in` , `!in` and `between` , and `!between` operators respectively. The `in` and `!in` operators are intended to select a subset of allowed values of drop-down fields, for example, **Status** in Jira or **State** in . Here is an example of copy and update selectors in a section of a configuration file that limit a copy and update service from Jira to to defects with a Status within a certain subset of allowed values:

```
<CopySelectors>
    <CopySelector>Status in To Do, In Progress</CopySelector>
</CopySelectors>
<UpdateSelectors>
    <UpdateSelector>Status in To Do, In Progress</UpdateSelector>
```

```
</UpdateSelectors>
```

In the following example, the defects are selected using the `!in` operator. The returned subset is effectively the same as the subset of defects limited by the `in` operator in the example above. Note that when using negation with drop-down fields that allow empty values like Priority, the subset of included work items will contain work items where this field is empty. If inclusion of those items is undesirable, the `in` operator should be preferred. Empty field values are not supported by subset selectors.

```
<CopySelectors>
    <CopySelector>Status !in In Review,Done</CopySelector>
</CopySelectors>
<UpdateSelectors>
    <UpdateSelector>Status !in In Review, Done</UpdateSelector>
</UpdateSelectors>
```

The `between` and `!between` operators select work items where a specific field falls within a certain range, where both ends of the range are included. The range selectors are intended to work with date fields such as **Created** in Jira or **CreationDate** in . Here is an example of copy and update selectors in a section of a configuration file that limit a copy and update service from to Jira to stories created within a certain date range:

```
<CopySelectors>
    <CopySelector>CreationDate between 2016-07-05 and 2016-07-06</CopySelector>
</CopySelectors>
<UpdateSelectors>
    <UpdateSelector>CreationDate between 2016-07-05 and 2016-07-06</UpdateSelector>
  </UpdateSelectors>
```

A variation of the range selector that uses the `!between` operator will select only stories that were created on either side of the date range:

```
<CopySelectors>
    <CopySelector>CreationDate !between 2016-07-05 and 2016-07-06</CopySelector>
</CopySelectors>
<UpdateSelectors>
    <UpdateSelector>CreationDate !between 2016-07-05 AND 2016-07-06</UpdateSelector>
</UpdateSelectors>
```

Multiple Configuration Files

If you want to do any of the following, setting up multiple configuration files is recommended:

- Map to more than one workspace in
- Map multiple artifact types
- Map to multiple containers (such as domains or projects in Quality Center) in the other system

To run multiple instances of the connector for different configuration files, pass in the list of configuration files as arguments to the `ca_agile_central2_jira_connector.rb` script or `ca_agile_central2_jira_connector.exe`.

```
ca_agile_central2_jira_connector.rb config_workspaceA.xml config_workspaceB.xml
```

The connector processes the configuration files based on the command line argument order and processes one file at time.

Once it processes every configuration file, the connector will sleep based on the sleep value. The default is 15 minutes. To change the sleep value between runs, set this value (in minutes) as the last command line argument. We recommend a sleep value of 10 minutes or more and advise against anything less.

```
ca_agile_central2_jira_connector.rb  config_workspaceA.xml  config_workspaceB.xml  15
```

XML Tags in the Jira Connector

The following sections contain definitions of the XML tags in each section of the configuration file.

NOTE
Avoid special characters contained in a workspace or project name that are markup-sensitive.

Some examples of markup sensitive characters are:

- & ampersand becomes &
- > greater than becomes >
- < less than becomes <

Example: Research & Development becomes Research & Development

XML Tags in the <RallyConnection> Section

<RallyConnection> Required. Opening parent tag for this section.

<Url> Required. Server used to connect to (excluding the HTTPS prefix, as the connector adds https://).

Sample values:

- `sandbox.rallydev.com`
- `rally1.rallydev.com`
- `myRally.mydomain.com`
- `192.168.23.24`

<User> Required if no APIKey specified. Login name for user to make the Web Services requests to create or update work items in .

Sample value: `user@company.com`

<Password> Required if no APIKey specified. Password for user to make the Web Services requests to create or update work items in . **Note:** The first time the connector runs, it will encode the password so it is not saved in plain text.

Sample value: `mypassword`

<APIKey> APIKey (from Application Manager) for the account to be used to access from the connector. If this is specified, you do not need to supply a username and password. You do not need to add the user associated with the given APIKey value to the list of allowed users.

Sample value: `ABC123plmokn43315...`

<WorkspaceName> Required. Workspace in you want to copy and update work items.

Sample values:

- `My Workspace`
- `MyWorkspace`

<Projects> Required. Contains a list of project tags. Each tag refers to one project that will be used when finding new work items to copy to the other system. For updating work items from to the other system, all projects in `<WorkspaceName>` are considered. At least one project must be specified in this tag.

Sample value: `<Project>Rally1</Project> <Project>Rally2</Project>`

<ArtifactType> Required. Type of artifact you want to create or update in .

Sample values:

- `(Defect, defect), (Story, UserStory, HierarchicalRequirement), TestCase`

<ExternalIDField>

Required. custom string field (name and display name must be identical) that stores the unique ID for the other system. Refer to the Create an External ID field in section.

Sample value: `JiraID`

<SuppressDeprecationWarning>

Changes the WARN message about WSAPI version deprecation in the logfile to an INFO message.

<CrosslinkUrlField>

custom field of type string (not-selectable) or text (selectable) containing a hyperlink to the corresponding bug in Jira. Only populated during a copy.

Sample value: `JiraLink`

<CopySelectors>

Criteria to use when finding new issues to copy to Jira. An individual expression has the form `<field> <relation> <value>`, where:

- `<field>` is the name of a artifact field
- `<relation>` is one of `=, !=, gt, lt, gte, lte`
- `<value>` is a valid value for the `<field>`

You can have multiple CopySelector criteria within the CopySelectors. All given CopySelector conditions are ANDed together.

```
<CopySelectors>
   <CopySelector>
      expression
   </CopySelector>
   ....
</CopySelectors>
```

Sample value: `<CopySelector>State = Open</CopySelector>`

<UpdateSelectors>

Criteria to use when finding existing issues in that should be updated to Jira. An individual selector specification has the form <field> <relation> <value>, where:

- `<field>` is the name of a artifact field
- `<relation>` is one of `=, !=, gt, lt, gte, lte`
- `<value>` is a valid value for the `<field>`

You can have multiple `<UpdateSelector>` criteria within `<UpdateSelectors>`. All given `<UpdateSelector>` conditions are ANDed together.

```
<UpdateSelectors>
   <UpdateSelector>
      expression
   </UpdateSelector>
   ....
</UpdateSelectors>
```

Sample value: `<UpdateSelector>Release != alpha</UpdateSelector>`

<FieldDefaults>

Used when the destination system has a required field, but the field is not going to be mapped in the configuration file. This will cause field F1 to be set to value of V1.

```
<FieldDefaults>
   <Field>
      <Name>F1</Name>
      <Default>V1</Default>
   </Field>
   [....]
</FieldDefaults>
```

Learn more in Map Required Fields.

</RallyConnection>	Required. Closing parent tag for this section.

XML Tags in the <JiraConnection> Section

<JiraConnection>	Required. Opening parent tag for this section.
<Url>	Required. Jira server name (or IP address) and port. Syntax is server:port. Sample value: `myjiraserver:8080`
<User>	Required. The Jira username. Must have permissions for the following operations: Browse Projects, Create Issues, Edit Issues, Schedule Issues, Assign Issues, Assignable User, Resolve Issues, Close Issues, Modify Reporter, Add Comments, Create Attachments. In most cases, a Jira user in the jira-developers group has these permissions. Sample value: `jirauser`
<Password>	Required. Password for the Jira user. The first time the connector runs, it will encode the password so it is not saved in plain text. If the `<Url>` value indicates that a Jira OnDemand instance is being accessed (with a distinctive `*.atlassian.net` value) then the value for the Password tag *must* be an API token value. The token value is generated from within the Jira tool after navigating to the **Account Settings, Security** page. Sample values: `mypassword`
<AccountID>	Jira user AccountID. This tag *must* be used if the `<Url>` has a value that indicates that a Jira OnDemand site is the target. A Url of this sort contains `.atlassian.net` The user's Jira account ID value can be obtained by observing the browser status line after selecting the user's icon (lower left) and hovering over the 'Profile'. Sample value: `abcde12345RST654CVB369CDE89`
<ProxyURL>	Jira field if using a proxy. Sample value: `proxy_host:port`
<ProxyUser>	Jira field that may be used depending on the setup of the proxy authentication. Sample value: `proxy_user_name`
<ProxyPassword>	Jira field that may be used depending on the setup of the proxy authentication. Sample value: `proxy_password`
<ArtifactType>	Required. Jira issue type for the issues you want to create or update in Jira. Sample values: Bug, Task, Initiative, Defect, User Story, ...
<IDField>	Jira field used to store the unique ID of an issue, defaults to *key*. The ID field can also be used as the value for this tag. Using the value of ID ensures that the connector can locate an issue even if the project for the issue is changed. When an issue is moved to another project in Jira the key value will change but the ID value does not. Sample value: `key`
<ExternalIDField>	Required. Jira custom field (Number Field type) that stores the unique ID for a Rally work item. For more information, refer to Create an External ID field in Jira. Sample value: `RallyID`
<ExternalEnd UserIDField>	A custom field in Jira of type Text Field (single line) which will be used to store the FormattedID of the work item (such as DExx). Sample value: `RallyFormattedID`
<CrosslinkUrlField>	Jira custom field (URL Field or Free Text types) containing text of a hyperlink to the corresponding artifact in . Only populated during Copy. Sample value: `RallyURL`

<Project>

Required. Specify the Jira Project Key name which contains the issues to be associated with work items. Be sure to specify the Jira Project Key and not the Jira Project Name. The Project Key is a string of uppercase characters entered as the key when the project was created.

Sample value: `TST`

<CopySelectors>

Criteria to use when finding new Jira issues to copy to . An individual selector specification has the form: `<field> <relation> <value>, where:`

- `<field>` is the name of a Jira issue field
- `<relation>` is one of `=, !=, gt, lt, gte, lte`
- `<value>` is a valid value for the `<field>`.

You can have multiple CopySelector criteria within the CopySelectors. All given CopySelector conditions are ANDed together.

```
<CopySelectors>
   <CopySelector>
      expression
   </CopySelector>
   ....
</CopySelectors>
```

Sample values:

- `<CopySelector>Status != Closed</CopySelector>`
- `<CopySelector>Priority = Low</CopySelector>`

<UpdateSelectors>

Criteria to use when finding existing issues in Jira that should be updated to . An individual selector specification has the form `<field> <relation> <value>` , where:

- `<field>` is the name of a Jira issue field
- `<relation>` is one of `=, !=, gt, lt, gte, lte`
- `<value>` is a valid value for the `<field>`.

You can have multiple UpdateSelector criteria within the UpdateSelectors. All given UpdateSelector conditions are ANDed together.

```
<UpdateSelectors>
   <UpdateSelector>
      expression
   </UpdateSelector>
   ....
</UpdateSelectors>
```

Sample value: `<UpdateSelector>Release != alpha</UpdateSelector>`

<WorkflowFile>

The relative pathname, from the current working directory, to the workflow XML file. See Configure Jira workflow and Configure A More Restrictive Custom Workflow.

```
<WorkflowFile>
   filename
</WorkflowFile>
```

Sample value: `../../JIRA/jira_workflow.xml`

<FieldDefaults>

Used when the destination system has a required field, but the field is not going to be mapped in the configuration file. This will cause field F1 to be set to value of V1.

```
<FieldDefaults>
   <Field>
      <Name>F1</Name>
      <Default>V1</Default>
   </Field>
```

```
    [....]
</FieldDefaults>
```

Learn more in Map Required Fields.

<OpenTimeout> Maximum time in seconds for which the connector will wait in an attempt to open a connection to the Jira server. Maximum value permitted is 999.

```
<OpenTimeout>
   seconds
</OpenTimeout>
```

Sample values:

- `180`
- `60`

<ReadTimeout> Maximum time in seconds for which the connector will wait to receive a read block of information from the Jira server. Maximum value permitted is 999.

```
<ReadTimeout>
   seconds
</ReadTimeout>
```

Sample values:

- `60`
- `180`

</JiraConnection> Required. Closing parent tag for this section.

XML Tags in the <Connector> Section

<Connector> Required. Opening parent tag for this section.

<FieldMapping> Required. Used to specify what fields are to be mapped between the two systems.

```
<FieldMapping>
      ....
</FieldMapping>
```

See Field Mapping.

<RallyFieldHandlers> (opening tag)

<RallyConcat FieldHandler> Using the contents of `Dest-fieldname` as the initial string, appends each additional Src-fieldnameN to Dest-fieldname, using HTML break tags (<br />) to separate each additional field value.

```
<RallyConcatFieldHandler>
     <FieldName>
        Dest-fieldname
     </FieldName>
     <ConcatFields>
        <Field>Src-fieldname1</Field>
        <Field>Src-fieldname2</Field>
        <Field>.....</Field>
     </ConcatFields>
</RallyConcatFieldHandler>
```

Sample values:

- *Some custom field*
  Description
  some other field

.....

<RallyDateTimeFieldHandler>

Bi-directional. to Other: Convert the ISO date in field F1 into a date string formatted as specified by the string S1 (as per Ruby's DateTime.strftime function) and store this new date string into the field F1 is mapped to.

```
<RallyDateTimeFieldHandler>
    <FieldName>F1</FieldName>
    <DateTimeFormat>S1</...>
</RallyDateTimeFieldHandler>
```

Sample value:

- CreationDate (being mapped to Due Date)
  %Y-%m-%d

<RallyUserFieldHandler>

Used to map usernames. See Map user names.

```
<RallyUserFieldHandler>
    <FieldName>
       name-of-object
    </FieldName>
    <ReferencedFieldLookupID>
       name-of-field
    </ReferencedFieldLookupID>
</RallyUserFieldHandler>
```

Sample values:

- `User,Owner,Tester`
- `Name,FormattedID`

<RallyEnumFieldHandler>

Allows for the mapping of non-alike, drop-down values between the two systems. See Map Drop-Down Values.

```
<RallyEnumFieldHandler>
   ....
</RallyEnumFieldHandler>
```

<RallyReferenceFieldHandler>

Used to map a field from an object. Direction: Jira to only. If no TestFolder is specified, then ALL tests in the TestPlan will be copied. If a TestFolder is specified, then ALL Tests in that folder only will be copied (but not subfolders or Tests in subfolders).

```
<RallyReferenceFieldHandler>
   <FieldName>
      name-of-field
   </FieldName>

   <ReferencedFieldLookupID>
      name-of-field
   </ReferencedFieldLookupID>
</RallyReferenceFieldHandler>
```

Sample values:

- TestCase, Parent, TestCaseResult, WorkProduct, TestSet, TestFolder, Requirement
- ObjectID, FormattedID

<RallyCSVUserMappingFieldHandler>

Specifies the name of a field being mapped (name-of-field) and a CSV file where user mappings are stored (name-of-file). The CSV file is used to lookup and transform usernames based on the mappings specified within. See a more detailed explanation on the FAQ page or an example XML file.

```
<RallyCSVUserMappingFieldHandler>
```

```
      <FieldName>
         name-of-field
      </FieldName>
      <FileName>
         name-of-file
      </FileName>
   </RallyCSVUserMappingFieldHandler>
```

Sample values:

- SubmittedBy
  MyUserMappings.csv

<RallyKeyword2TagFieldHandler>

Specifies the mapping of Tags to Jira labels. See example configuration file at JIRArest-Labels-to-Tags.pxml.

```
   <RallyKeyword2TagFieldHandler>
      <Type>
         type-value
      </Type>
      <Delimiter>
         some-character
      </Delimiter>
   </RallyKeyword2TagFieldHandler>
```

Sample values:

- String
  ,

<RallyBooleanFieldHandler>

Specifies the mapping of Boolean fields to Jira Radio buttons only. Works bi-directionally. See a more detailed explanation on the FAQ page.

```
   <RallyBooleanFieldHandler>
      <FieldName>
         name-of-Rally-field
      </FieldName>
      <Mappings>
         <Field><Rally>false</Rally>
         <Other>Jira-field-value</Other></Field>
         <Field><Rally>true</Rally>
         <Other>Jira-field-value</Other></Field>
      </Mappings>
   </RallyBooleanFieldHandler>
```

</RallyFieldHandlers>

(closing tag)

<OtherFieldHandlers>

(opening tag)

<OtherEnumFieldHandler>

Allows a user to map non-alike, drop-down values between the two systems. See Map Drop-Down Values.

```
   <OtherEnumFieldHandler>
      ....
   <OtherEnumFieldHandler>
```

<OtherConditionalEnumFieldHandler>

Very similar to the **<OtherEnumFieldHandler>** except with this one only the exceptions need to be declared. Corresponding entries in the pull-down lists of both system which are identical, need not be declared.

```
   <OtherConditionalEnumFieldHandler>
      ....
   </OtherConditionalEnumFieldHandler>
```

<OtherConcatFieldHandler>

Using the contents of `Dest-field` as the initial string, appends each additional `Src-fieldN` to `Dest-field` , using HTML break tags (<br />) to separate each additional field value.

```
<OtherConcatFieldHandler>
   <FieldName>
      Dest-field
   </FieldName>
   <ConcatFields>
      <Field>Src-field1</Field>
      <Field>Src-field2</Field>
      <Field>.....</Field>
   </ConcatFields>
</OtherConcatFieldHandler>
```

Sample values:

- *Some custom field*
  Description
  Some other custom field
  .....

<JiraNewlineFieldHandler>

Used to declare the Jira Description field is to be processed, having newlines converted. Currently, only the Description field is supported.

```
<JiraNewlineFieldHandler>
   <FieldName>
      Description
   </FieldName>
</JiraNewlineFieldHandler>
```

<JiraUserToEmailFieldHandler>

Used to declare the Jira field to be used for finding an email address. This can be used to map the Jira Reporter email field or Jira Assignee email field to the Owner field when sending information from Jira to . See example below.

```
<JiraUserToEmailFieldHandler>
   <FieldName>
      name-of-field
   </FieldName>
</JiraUserToEmailFieldHandler>
```

Sample values:

- (opening tag)
  Reporter
  (closing tag)

</OtherFieldHandlers> (closing tag)

<RelatedObjectLinkers> For more complex mappings of fields (like collections).

<RallyAttachmentLinker /> Used to copy attachments between Jira and (bi-directional). Can be used on any object which supports attachments (in : tasks, defects, stories, and tests; in Jira: any issue type with attachments). See Jira-config-bugAttachment2Rally.pxml for an example.

<RallyJiraCommentLinker /> Used to copy discussions or comments between and Jira.

WARNING
There is a known limitation with this feature. Once discussions have been synced with Jira comments, they should never be modified in Jira. It will cause a recursive copy, back-and-forth between the two systems, with the discussions and comments growing with each iteration. This occurs because the connector compares the two sets of discussions and comments. If they are not identical, it assumes they need to be re-synced. However, discussions are read-only (once they have been created), so an edited comment in Jira is copied to as a new discussion. At that

point the discussions and comments between the two system will be out of sync again, and this loop will continue indefinitely. See Jira-config-RallyJiraCommentLinker.pxml for an example.

</RelatedObjectLinkers> (closing tag)

</Connector> Required. Closing parent tag for this section.

XML Tags in the <ConnectorRunner> Section

<ConnectorRunner> Required. Opening tag for this section.

<Preview> Allows you to enable a preview mode for testing where NO objects are copied or updated in either system.

Sample values:

- `false` (default)
- `true`

<LogLevel> Determines what type of messages are written to the log file. The highest level is debug where all messages are displayed. The default log level is Info.

Sample values:

- Fatal, Error, Warning, Info, Debug

<Services> Required. Use the copy services to initially reflect items between systems, then use the update services to keep those reflected items up to date.

The alternative update service searches first for updates and pushes only the recently changed fields to the other system. It then searches the other system for updates and pushes all mapped fields into . This allows changes done in both systems to not cause overwrites of old data.

Sample values:

- Copy services:
  - `COPY_JIRA_TO_RALLY`
  - `COPY_RALLY_TO_JIRA`
- Update services:
  - `UPDATE_JIRA_TO_RALLY`
  - `UPDATE_RALLY_TO_JIRA`
- Alternative update service:
  - `UPDATE_RALLYFIELDS_AND_OTHER` (may not be used in conjunction with the other update services)
- See FAQ for more information.

<Emailer> The Emailer feature with NO Authorization. Defines which logfile messages to email. To send email to multiple emails, use a semi-colon to delimit each email address.

> **NOTE**
> Versions of the connector prior to 3.2.0 used the sub-tag `<Level>` instead of `<LogLevel>` in the Emailer block. While we recommend against it, you can still use the `<Level>` sub-tag but you must invoke the connector with the `--run-on-validation-failure` or `-r` flag in order for the connector to work. We recommend that you update your configuration file to use the `<LogLevel>` tag to gain the benefit of running the XML configuration file validation.

```
<Emailer>
   <LogLevel>value</LogLevel>
   <SendEmailFrom>value</SendEmailFrom>
   <SendEmailTo>value</SendEmailTo>
   <SMTPServer>value</SMTPServer>
   <SMTPPort>value</SMTPPort>
</Emailer>
```

Sample value: See example.

<Emailer> The Emailer feature with Authorization. Defines which logfile messages to email.

> **NOTE**
> Versions of the connector prior to 3.2.0 used the sub-tag `<Level>` instead of `<LogLevel>` in the Emailer block. While we recommend against it, you can still use the `<Level>` sub-tag but you must invoke the connector with the `--run-on-validation-failure` or `-r` flag in order for the connector to work. We recommend that you update your configuration file to use the `<LogLevel>` tag to gain the benefit of running the XML configuration file validation.

```
<Emailer>
   <LogLevel>value</LogLevel>
   <SendEmailFrom>value</SendEmailFrom>
   <SendEmailTo>value</SendEmailTo>
   <SMTPServer>value</SMTPServer>
   <SMTPPort>value</SMTPPort>
   <SMTPDomain>value</SMTPDomain>
   <SMTPUser>value</SMTPUser>
   <SMTPPassword>value</SMTPPassword>
   <SMTPSec>value</SMTPSec>
</Emailer>
```

Sample value: See example.

</ConnectorRunner> Required. Closing parent tag for this section.

Run the Connector for Jira

To run the connector, use the executable, which is a result of installing the rallyeif-jira gem. This executable is not visible in the installation directory, it resides with the other contents of the gem located with all the Ruby gems associated with your Ruby installation.

The connector accepts these command line flags:

```
$ ca_agile_central2_jira_connector.rb  -h
        Usage: ca_agile_central2_jira_connector.rb <options> <config_filename> <interval_in_minutes>

        where <options> are:
          --version, -v:   Prints the revision number of the connector.
          --precheck, -p:   Precheck, the connector will run in preview mode.
          --run-on-validation-failure, -r:   Will ignore results of XSD Schema validation and run connector.
          --log-file-size, -s <n>:   Set the size limit for a log file to <n> megabytes.
          --max-log-files, -m <n>:   Set the maximum number of log files in log file rotation to <n>.
          --help, -h:   Print this Usage message.
          --cleartext, -c:   Bypass encrypting the credentials if they are already in clear text.

        where <config_filename>
          File system reference to an XML file specifying the connector configuration.  Usual extension is
 '.xml'.

        where <interval_in_minutes>
          The interval, in minutes, for which the connector will run after executing command the first time.
          -1 will only run the connector once and will exit afterwards.
```

Once the `jira_config.xml` is set up, you can start running the connector. To start the connector, use one of the following command lines:

```
ca_agile_central2_jira_connector.rb  jira_config.xml  -1
```

The `-1` interval value in the examples above specify to run the connector to process each service named in the `<Connector><Services>` once and then exit.

Optionally, you can add a number to the end of the command line that specifies the time in minutes the connector should sleep between successive executions of all services. The default is 15 minutes. We recommend sleep intervals of no less than 10 minutes. For example:

```
ca_agile_central2_jira_connector.rb  jira_config.xml  10
```

To stop the service, use Control-C in the command shell.

Credential Encryption for the Jira Connector

As of version 4.6.0 of the connector, credentials occurring in the *Connection sections of the configuration file that are in clear-text are encrypted after the initial run using that configuration file. Credentials are identified as any configuration item tag appearing in a *Connection section whose name is *User, or any *Password or APIKey.

Credentials in the Emailer section (SMTPUser and SMTPPassword) will still be encoded. The credential encryption is localized for the installation of the connector and various aspects of the configuration file.

In practice, as long as the installation and execution folders are constant and the key parameters in the configuration do not change, you are free to modify most of the contents of the configuration file without having to reset the credentials to their clear-text values. However, care must be taken in environments that do not have such stable characteristics to frequently check the log files for any entries that indicate environmental changes have taken place that prevent the connector from decrypting the encrypted values for use in the connector execution.

The `-c` option can be used to specify leaving the password in clear case (if it is already in clear text) instead of encrypting. The command line for running the connector is:

```
ca_agile_central2_jira_connector.exe -c configfile.xml
```

The `-1` option may still be used at the end of the command if needed.

Modify the Logging Parameters for the Jira Connector

The connector logs messages as it is processing into a file named `rallylog.log` in the current working directory. By default, the maximum size of a log file is 5 MB and the log rotation is limited to ten files. You can adjust the maximum size of the log file and adjust the maximum count of log files in the log rotation by specifying command line arguments as follows:

The `-s <integer>` option pair can be used to specify the maximum size of the log file in MB increments (up to 100 MB). This can also be expressed as `log-file-size <integer>`.

The `-m <integer>` option pair can be used to specify the maximum number of files in the log file rotation scheme. This can also be expressed as `--max-log-files <integer>`.

Example: To set the log file maximum size to 50 MB and the maximum log file rotation count to 20 files for a single invocation of the connector:

```
ca_agile_central2_jira_connector.rb -s 50 -m 20 xxx_config.xml -1
```

OR

```
ca_agile_central2_jira_connector.rb --log-file-size 50 --max-log-files 20 xxx_config.xml -1
```

Jira Connector Workflows

The connector installation includes four workflow xml files, located in the root of the connector's directory.

```
jira_issue_workflow.xml
jira_workflow.xml
simplified_issue_workflow.xml
simplified_workflow.xml
```

`jira_issue_workflow.xml` and `simplified_issue_workflow.xml` are intended for non-bug issue types. `jira_workflow.xml` and `simplified_workflow.xml` are intended for bugs. `simplified_workflow.xml` and `simplified_issue_workflow.xml` are intended to work in Jira projects that use Jira 7 default Software Simplified Workflow. `jira_workflow.xml` and `jira_issue_workflow.xml` are intended to work in Jira projects that use Jira 7 default Jira Workflow.

Three scenario examples are described:

- When a Jira project is associated with Software Simplified Workflow, use a simplified workflow file. `<WorkflowFile>` tag in `<JiraConnection>` section should reference `simplified_workflow.xml` if `<ArtifactType>` is bug, or `simplified_issue_workflow.xml` if `<ArtifactType>` is set to any other supported issue type:
  ```
  <WorkflowFile>simplified_workflow.xml</WorkflowFile>
  ```
- When a Jira project is associated with Jira Workflow, use a Jira workflow file. `<WorkflowFile>` tag in `<JiraConnection>` section should reference `jira_workflow.xml` if `<ArtifactType>` is bug, or `jira_issue_workflow.xml` if `<ArtifactType>` is set to any other supported issue type:
  ```
  <WorkflowFile>jira_workflow.xml</WorkflowFile>
  ```
- When a Jira project is associated with a custom workflow, you need to create a custom workflow file first, then reference it in the `<WorkflowFile>` tag in `<JiraConnection>` section, like this:
  ```
  <WorkflowFile>my_custom_workflow.xml</WorkflowFile>
  ```

Configure a Jira Custom Simplified Workflow

The connector does not work with workflow xml files directly exported from Jira. If your Jira project is not associated with either default Software Simplified Workflow or default Jira Workflow, workflow files that come with the connector cannot be used without customization.

Consider the following example:

A Jira 7 project is associated with a default Software Simplified Workflow. Your Jira administrator creates a new custom workflow and associates it with your Jira project. This custom workflow is very similar to the default simplified workflow with one modification: when issue is transitioned to Done status the Resolution is no longer automatically set to default Done value, but instead the transition to Done status now triggers Resolve Issue Screen where a user has to select Resolution.

The only modification you will need to make to `simplified_workflow.xml` file (included with the connector installation) is to add `<Field>resolution</Field>` nested inside each `<Action/>` tag. Here is the modified `simplified_workflow.xml` file compatible with this custom workflow:

```
<?xml version="1.0"?>
        <!-- Custom Simplified Workflow--> <Workflow>
        <Step RallyState="Submitted" JiraStatusName="To Do"> <Transition JiraStatusName="In Progress">
        <Action ID="11" Name="To Do"/> </Transition>
        <Transition JiraStatusName="In Review">
        <Action ID="11" Name="To Do"/>
        </Transition>
        <Transition JiraStatusName="Done">
        <Action ID="11" Name="To Do"/> </Transition>
        </Step>
        <Step RallyState="Open" JiraStatusName="In Progress"> <Transition JiraStatusName="To Do">
        <Action ID="21" Name="In Progress"/> </Transition>
        <Transition JiraStatusName="In Review">
```

```
<Action ID="21" Name="In Progress"/> </Transition>
<Transition JiraStatusName="Done"> <Action ID="21" Name="In Progress"/>
</Transition> </Step>
<Step RallyState="Fixed" JiraStatusName="In Review"> <Transition JiraStatusName="To Do">
<Action ID="31" Name="In Review"/> </Transition>
<Transition JiraStatusName="In Progress">
<Action ID="31" Name="In Review"/> </Transition>
<Transition JiraStatusName="Done">
<Action ID="31" Name="In Review"/> </Transition>
</Step>
<Step RallyState="Closed" JiraStatusName="Done"> <Transition JiraStatusName="To Do">
<Action ID="41" Name="Done"> <Field>resolution</Field>
</Action>
</Transition>
<Transition JiraStatusName="In Progress">
<Action ID="41" Name="Done"> <Field>resolution</Field>
</Action>
</Transition>
<Transition JiraStatusName="In Review">
<Action ID="41" Name="Done"> <Field>resolution</Field>
</Action> </Transition>
</Step> </Workflow>
```

This customized workflow makes it possible to map Resolution field from to Jira during COPY_RALLY_TO_OTHER and UPDATE_RALLY_TO_OTHER service (provided that Resolution is listed in the `<FieldMapping>` section and individual drop-down values are mapped in `<OtherEnumFieldHandler>` section. This `<OtherEnumFieldHandler>` is optional if Resolution drop-down values are identical between the two systems). On a related note, unlike the custom workflow example above, the default Software Simplified Workflow does not lend itself for setting Resolution field from to Jira during COPY_RALLY_TO_OTHER and UPDATE_RALLY_TO_OTHER service. While it is possible to set Resolution from to the Jira project associated with the default Software Simplified Workflow, it will require changes to the Jira project, specifically adding Resolution field to Software Development Bug Screen and to Software Development Default Issue Screen. Adding Resolution to those screens makes Resolution a required field for every status transition, no longer limited to transition to the final status. This approach is cumbersome and may be undesirable as far as Jira best practices are concerned.

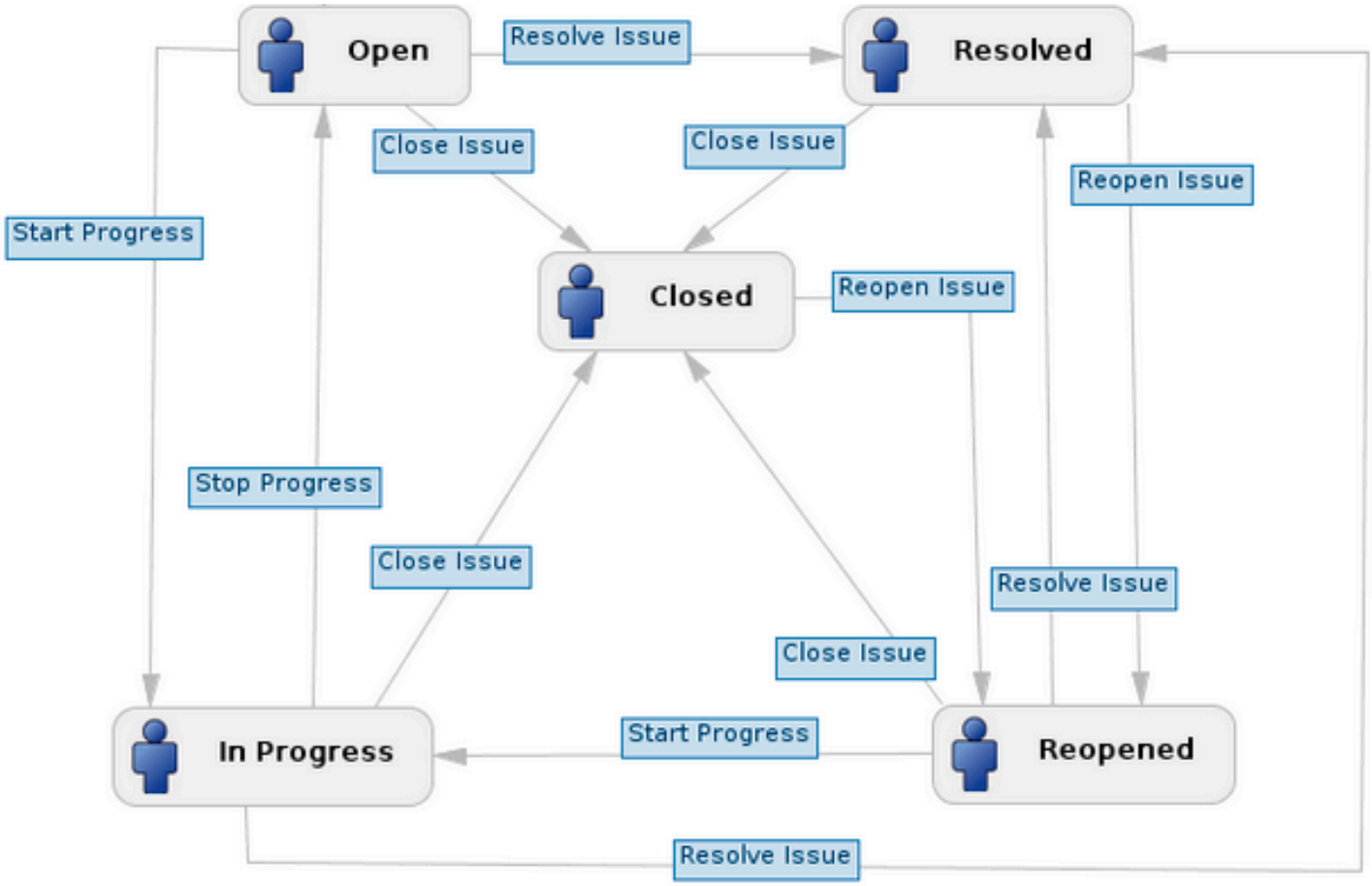
Configure a Jira More Restrictive Custom Workflow

In order to support updates from to Jira in the case where a custom Jira defect workflow has been defined, the connector for Jira allows you to describe the custom workflow in an XML file. Using this workflow definition, the connector can transition a Jira issue through the workflow until its status matches the state of a defect.

As an example, consider the default workflow for the Jira issue type of bug pictured below. Assume there is a defect in DE1 that was initially created in Jira (as issue TST-1), marked in Jira as Resolved (Fixed), and some time later copied to , where it was assigned a state of Fixed.

Subsequently, a developer discovers that the defect is not fixed, and updates its state to Open. Since in the Jira workflow there is no transition from the Resolved status to the In Progress status (to match the state

Open), the issue must go through the Reopen Issue and Start Progress transitions to get to the desired



status.

The XML file (`workflow.xml`) that describes the default Jira defect workflow is shown below:

```
<Workflow>
        <Step RallyState="Submitted" JiraStatusName="Open">
                <Transition JiraStatusName="In Progress">
                        <Action ID="301" Name="Stop Progress"/>
                </Transition>
                <Transition JiraStatusName="Resolved">
                        <Action ID="3" Name="Reopen Issue"/>
                        <Action ID="4" Name="Start Progress"/>
                        <Action ID="301" Name="Stop Progress"/>
                </Transition>
                <Transition JiraStatusName="Closed">
                        <Action ID="3" Name="Reopen Issue"/>
                        <Action ID="4" Name="Start Progress"/>
                        <Action ID="301" Name="Stop Progress"/>
                </Transition>
                <Transition JiraStatusName="Reopened">
                        <Action ID="4" Name="Start Progress"/>
                        <Action ID="301" Name="Stop Progress"/>
                </Transition>
        </Step>
```

```
<Step RallyState="Open" JiraStatusName="In Progress">
        <Transition JiraStatusName="Open">
                <Action ID="4" Name="Start Progress"/>
        </Transition>
        <Transition JiraStatusName="Reopened">
                <Action ID="4" Name="Start Progress"/>
        </Transition>
        <Transition JiraStatusName="Resolved">
                <Action ID="3" Name="Reopen Issue"/>
                <Action ID="4" Name="Start Progress"/>
        </Transition>
        <Transition JiraStatusName="Closed">
                <Action ID="3" Name="Reopen Issue"/>
                <Action ID="4" Name="Start Progress"/>
        </Transition>
</Step>

<Step RallyState="Fixed" JiraStatusName="Resolved">
        <Transition JiraStatusName="Open">
                <Action ID="5" Name="Resolve Issue">
                        <Field>resolution</Field>
                </Action>
        </Transition>
        <Transition JiraStatusName="In Progress">
                <Action ID="5" Name="Resolve Issue">
                        <Field>resolution</Field>
                </Action>
        </Transition>
        <Transition JiraStatusName="Reopened">
                <Action ID="5" Name="Resolve Issue">
                        <Field>resolution</Field>
                </Action>
        </Transition>
        <Transition JiraStatusName="Closed">
                <Action ID="3" Name="Reopen Issue"/>
                <Action ID="5" Name="Resolve Issue">
                        <Field>resolution</Field>
                </Action>
        </Transition>
</Step>

<Step RallyState="Closed" JiraStatusName="Closed">
        <Transition JiraStatusName="Open">
                <Action ID="2" Name="Close Issue">
                        <Field>resolution</Field>
                </Action>
        </Transition>
        <Transition JiraStatusName="In Progress">
                <Action ID="2" Name="Close Issue">
                        <Field>resolution</Field>
                </Action>
        </Transition>
        <Transition JiraStatusName="Reopened">
```

```
                        <Action ID="2" Name="Close Issue">
                                <Field>resolution</Field>
                        </Action>
                </Transition>
                <Transition JiraStatusName="Resolved">
                        <Action ID="701" Name="Close Issue"/>
                </Transition>
        </Step>

        <Step RallyState="Reopened" JiraStatusName="Reopened">
                <Transition JiraStatusName="Open">
                        <Action ID="4" Name="Start Progress"/>
                        <Action ID="5" Name="Resolve Issue">
                                <Field>resolution</Field>
                        </Action>
                        <Action ID="3" Name="Reopen Issue"/>
                </Transition>
                <Transition JiraStatusName="In Progress">
                        <Action ID="5" Name="Resolve Issue">
                                <Field>resolution</Field>
                        </Action>
                        <Action ID="3" Name="Reopen Issue"/>
                </Transition>
                <Transition JiraStatusName="Resolved">
                        <Action ID="3" Name="Reopen Issue"/>
                </Transition>
                <Transition JiraStatusName="Closed">
                        <Action ID="3" Name="Reopen Issue"/>
                </Transition>
        </Step>
</Workflow>
```

Use the following procedure to create the XML workflow specification (refer to the default workflow XML above as an example).

1. Identify the steps in your Jira custom workflow.

   NOTE
   The process of creating the custom workflow XML is much easier if you create a diagram similar to the one above showing the Jira steps or statuses, corresponding states, and the transitions between them, including status and action IDs.

   a. In Jira, select the Administrator tab and select **Workflows** under the Global Settings section in the navigation pane on the left.
   b. Find the custom workflow and select the **Steps** link in the Operations column to display the View Workflow Steps screen.
      The `<Step>` sections in `workflow.xml` correspond directly with the steps in Jira.
2. Create states for each Jira status.
   For the Linked Status in each Jira step, find or create a corresponding state in . To create new states in , follow a procedure similar to that described under in Set Up States and Resolutions.
3. In a new file, create the beginning and ending XML tags:
   `<Workflow></Workflow>`
4. Inside these tags, create `<Step></Step>` blocks for each pair of corresponding and Jira state or statuses.
   The values of the `<Step>` tag should look like this:

```
<Step RallyState="Reopened" JiraStatusName="Reopened">
```

5. Create `<Transition>` blocks for each `<Step>` .

   Inside each `<Step>` block, create `<Transition>` blocks for each of the other state or status pairs. Each `<Transition>` under a `<Step>` describes how to get *from* another state or status *to* the one identified in the `<Step>` block you are working on. For example, under the Reopened step from the example above, there is a transition that describes how to get to Reopened from Jira status Open:

   ```
   <Step RallyState="Reopened" JiraStatusName="Reopened">
                   <Transition JiraStatusName="Open">
   ```

   Again, the status names can be found on the View Workflow Steps screen in Jira. For a custom workflow with N states, each `<Step>` block should contain N-1 `<Transition>` blocks in order to give the connector enough information to successfully update Jira status with any arbitrary state change in . It is not necessary to specify a transition from a state or status to itself.

6. Define `<Action>` blocks for each `<Transition>` .

   Inside each `<Transition>` block, create `<Action>` blocks for each action needed to progress the defect through the workflow:

   – From the status identified in the JiraStatusName value of the `<Transition>` block
   – To the status identified in the JiraStatusName value of the enclosing `<Step>` block

   Continuing with this example, to get from status Open to status Reopened, you need three actions: Start Progress, Resolve, and Reopen. The corresponding XML looks like this:

   ```
   <Step RallyState="Reopened" JiraStatusName="Reopened">
           <Transition JiraStatusName="Open">
                   <Action ID="4" Name="Start Progress"/>
                   <Action ID="5" Name="Resolve Issue">
                           <Field>resolution</Field>
                   </Action>
                   <Action ID="3" Name="Reopen Issue"/>
           </Transition>
   </Step>
   ```

   The action IDs can be found on the Jira View Workflow Steps page, in the Transitions column. As noted above, a diagram of the custom workflow is indispensable in this part of the process.

7. Define any fields that must be filled in for an `<Action>` .

   Notice in the example above that the Resolve Issue action has a `<Field>` element which indicates that the Resolution field should be supplied to Jira. The value of the Resolution field is the value of Resolution in the defect, after any transformations specified by an `<OtherEnumFieldHandler>` for Resolution (if the Resolution lists in and Jira do not match exactly). If selecting a link under Available Workflow Actions in the Jira user interface would result in a Jira screen requiring you to fill in a field in order to complete the action, you should mimic that process by providing the fields and values here.

Jira Connector Best Practices

The topics in this section describe best practices for you to follow related to the connector.

Use the <JiraUserToEmailFieldHandler>

```
<Config>
      ....
      <Connector>
        <FieldMapping>
              <Field> <Rally>Name</Rally>  <Other>Summary</Other>  </Field>
              <Field> <Rally>Owner</Rally> <Other>Reporter</Other> </Field>
              ....
        </FieldMapping>
```

```
    <OtherFieldHandlers>
          <JiraUserToEmailFieldHandler>
            <FieldName>Reporter</FieldName>
          </JiraUserToEmailFieldHandler>
    </OtherFieldHandlers>
  </Connector>
  ....
```

Mapping

- Mapping Jira Attachments

  NOTE

  attachments are limited to 50 MB. If a Jira attachment size is larger than 50 MB, it will not be copied from Jira to . The `<RallyAttachmentLinker>` supports bi-directional mapping between Jira and .

  Copy attachments between and Jira with the `<RallyAttachmentLinker>` option in the configuration file as follows:

```
<Config>
      ....
      <Connector>
           ....
        <RelatedObjectLinkers>
                      <RallyAttachmentLinker />
        </RelatedObjectLinkers>
        ....
```

  Known issue: DE23963 - If adding attachments to a child project and the connector user only has editor permissions to , the user must have at least editor permissions to all parent projects of the child project to add an attachment.

  See Jira-config-bugAttachment2Rally.pxml for an example.

- Map the Jira Sprint Field

  You can map to this field, but due to limitations in the Atlassian Jira REST API implementation, the connector can only support the mapping in the Jira-to- direction. Also, note that the value which displays in the target field will be the Jira Sprint field ID value, not the Jira Sprint field label value. In the future, if Atlassian adjusts their REST API for Jira to offer support for setting and updating Jira Sprint field values correctly under most conditions, bi-directional support for the Sprint field should be possible.

  An example specifying the supported directionality for the Sprint field:

```
<Config>
        ....
        <Connector>
                <FieldMapping>
                        ....
                        <Field>
                                <Rally>Iteration</Rally>
                                <Other>Sprint</Other>
                                <Direction>TO_RALLY</Direction>
                        </Field>
                        ....
                </FieldMapping>
```

- Map projects

  If you are copying issues from Jira to using the `<RallyReferenceFieldHandler>` with projects, you may see a *Could not find Project with name* error in the log file if the connector user does not have editor permissions to that project in .

NOTE
RallyReferenceFieldHandlers are for the most part deprecated. Prior to the 4.4.2 release, if multiple projects were specified in the configuration file Release or Iteration value was sometimes mapped incorrectly. By relocating some logic, that issue has been addressed as well as being able to handle the most common Reference fields (Project, Release, Iteration) automatically without needing explicit notation in the configuration file. If the connector detects that are using a RallyReferenceFieldHandler for one of those aforementioned fields, the connector will emit a deprecation warning message into the log file and will not register the field handler as that logic is now handled automatically. The are some edge cases where the use of RallyReferenceFieldHandler may be necessary, for fields such as Parent, Child, WorkProject, Owner, and other similar fields.

Known Limitations of the Jira Connector

LDAP not supported	The Jira connector does not support LDAP.
Artifact hierarchy	Artifact hierarchy is not maintained when copying/updating artifacts between and Jira.

Troubleshoot the Jira Connector

The following items describe how to resolve some problems you might have related to the connector.

See Log File for Messages

Once the connector is running, all errors are written to a log file in the current working directory where the connector was invoked. Informational messages, warnings, and errors are written to the log file depending on the value of the `<LogLevel>` tag in the `<ConnectorRunner>` section of the configuration file.

Connector Validation

Before the connector starts mapping objects between the two systems, it performs a validation that confirms:

- Connecting to is successful
- Connecting to the Jira system is successful
- Fields in the field mapping exist in
- Each field handler has a corresponding field mapping section in the configuration file

To confirm the validation is successful without moving objects between the two systems, the `<Preview>` tag in the configuration file may be set to `true` . This helps to experiment with different configuration options while debugging an issue.

Jira Custom Field Precaution (ExternalID) Field Default Value

The copy operation for the COPY_JIRA_TO_RALLY service is driven by the value in the custom field set up in Jira to hold the identifier for the corresponding artifact in . This field is often referred to as the ExternalID. Precautions:

- This Jira custom field must be defined as a type Number Field (not a string and not text) and must be defined with a default value of **-1** (minus 1).
- Old Jira items that existed before this Jira custom field was created will not have the default value of **-1** (as required), but instead it will be NULL. Because of this, the Jira query (performed by the connector) to identify items eligible to be copied to will not find these items. Should you want to include these existing items in the copy, perform a bulk change operation in Jira (typically found under Tools, Bulk Change after doing a search and filter) to set the value of the ExternalID custom field to a **-1** for items whose current value is NULL.

Jira Custom Field Precaution (ExternalID) Field Type

When the connector searches Jira for new artifacts, it attempts to find all Jira artifacts where the custom field ExternalID (or whatever you named it) is equal to a -1. For this reason, the field must be defined as a type Number Field. If it is defined as a Text Field (single line), you will see an error like:

```
        ERROR : RallyEIF::WRK::JiraConnection.rescue in find_new - Trying to find_new
via query: ....
        ERROR : RallyEIF::WRK::JiraConnection.initialize - ["The operator '=' is not
supported by the 'RallyObjectID' field."]
       INFO : RallyEIF::WRK::JiraConnection.disconnect - Disconnected from Jira
```

Jira Custom Field Precaution (General)

When defining a Jira custom field (for example, to receive some field from), the valid screens must be declared during this process. It is best to select ALL screens for the field.

If the Default Screen is not selected, and the field is declared Required, and a Field Default is supplied in the XML configuration file, you may see an error like this (in the following test, a Jira custom field named **Severity** was created as Required without the Default Screen checked):

```
ERROR : JiraConnection.initialize - Unrecognized field Severity, cannot continue
```

Unrecognized Field

When copying an artifact from to Jira, the following sample error can occur:

```
 INFO : RallyConnection.find_new -   Found 1 new defects in
 INFO : Connector.copy_to_other - Copy DE442 to Jira
DEBUG : Connector.block in map_fields_to_other -   Mapping Name(Test) - to -
 Summary(Test)
                ....
 WARN : JiraConnection.initialize - Unable to create new Jira Bug,
              Unrecognized field |RallyField|,
              unable to accurately realize postable data for this field in |create|
 mode
       ....
  ANY : ConnectorRunner.process_service - Finished Service COPY_RALLY_TO_JIRA
```

- A non-mapped field (such as a custom field like RallyLink or RallyFormattedID) is required, but not defined
- The custom field exists, but it is not visible to the specified `<Project>`

Field not Found

When copying an artifact from Jira to , the following sample error can occur:

```
DEBUG : RallyEIF::WRK::Connector.block in validate - RallyConnection field_mapping
 target "TestDateDeployment" existence validated
ERROR : RallyEIF::WRK::Connector.block in validate - FieldMapping: Other field
 "CustomDate Deploy" not found
ERROR : RallyEIF::WRK::ConnectorRunner.initialize - Invalid Configuration
```

- A mapped field (such as a custom field) is not visible on all screens

Not a Valid Field

Sometimes projects will not see custom fields if the custom fields existed before the project was created. The following error may occur for this issue:

```
ERROR : RallyEIF::WRK::JiraConnection.validate - Jira ExternalIDField name of
 "RallyID" is not a valid field name"
```

- Solution: Verify that all screens are selected for that custom field.

Mismatched External Identifier

This error can occur whenever artifacts are cloned or duplicated in either or Jira. If the artifact has already been copied to the other system, then the `<ExternalIDField>` has been given a value. When an artifact is cloned, the value in this field is carried to the new artifact. And then there will be two artifacts with the same value in this field.

This error also can occur when two different XML tags are incorrectly using the same custom field name (JiraID in the following example):

```
<Config>
        <RallyConnection>
                <Url>rally1.rallydev.com</Url>
                .....
                <ExternalIDField>JiraID</ExternalIDField>
                <CrosslinkUrlField>JiraID</CrosslinkUrlField>
        </RallyConnection>
        .....
```

The error message returned was:

```
DEBUG : RallyJest::JiraProxy.makeAnIssueInstance - Attempting to obtain a JiraIssue
 instance for RIT-37
DEBUG : RallyJest::JiraProxy.makeAnIssueInstance - Obtained JiraIssue for RIT-37
```

```
WARN : Connector.update_other - JIRA Bug RIT-37 matched to Rally defect DE27 has
mismatched external identifier in Rally item
 ANY : ConnectorRunner.process_service - Finished Service UPDATE_RALLY_TO_JIRA
```

Logfile is not Written to When Running the Connector as a .bat (Batch) Job

If the connector is being run as a batch job, verify that the user that owns or runs the batch job has access to the logfile directory.

is not Populated with Jira Artifact's URL

When creating a selectable link in the artifact (back to the Jira artifact), if the custom field in is not be completed, ensure the Type on the custom field is string or text, and not web link (see Create a CrosslinkUrlField in).

EnvironmentalKey ident_vector not Valid Error

The following error may be seen in the Jira connector version 4.6.0 and higher. If this error is seen, re-enter the userid and password for both Jira and .

```
.....
ERROR : Class.initialize - EnvironmentalKey ident_vector not valid for decryption
 target value
```

Using Parentheses in a Copy or Update Selector Value Results in Error in the JQL Query Error

When a selector contains a key/value pair that has a set of parentheses in its value, the connector does not format the JQL correctly. This is due to the way that the connector handles escaping the parentheses in a Copy/Update selector. This causes JQL parser to try and interpret the parentheses as a grouping construct instead of a value to search for and an error is thrown:

```
.....
DEBUG : RallyJest::JiraComm.block in execute_request - issuing a GET
 request for endpoint: /rest/api/2/search?jql=type="Bug" AND project =
 "Project" AND "RallyFormattedID" = -1 AND "CustomDropDown" = Value (New
 Value)&startAt=0&fields=*all
DEBUG : RallyJest::JiraComm.execute_request - {
"errorMessages": [
"Error in the JQL Query: Expecting ')' or ',' but got 'Value'. (line 1, character 90)"
],
"errors": {
```

The recommended workaround is to remove parentheses from the fields or use a different field for the copy/update selector.

Remove HTML from Description

When using the connector to copy data from to Jira, the Description field in Jira shows HTML tags. This is only an issue for Jira on-premises users.

In order to remove HTML from the Description field in Jira, a Jira administrator will need to enable wiki text rendering.

1. From the gear in the top right, select **Issues**.
2. Select **Field Configurations** on the left.
3. Select the name of the configuration you are using.
4. Scroll down to the Description field.
5. Select the Renderers link to the right of Description.
6. Select wiki style renderer from the dropdown
7. Select **Update**.

Adjust any Custom Field Handlers

If you have and use any custom field handlers (not the standard field handlers mentioned in the documentation) you must edit the `read_config` method so that the primary argument is `fh_info` (a Ruby Hash instance) and the code that examines that and assigns the `@field_name` is similar to the following:

```
fh_info.each do |name, value|
                  if name == 'FieldName'
                         @field_name = value
                  end
            end
```

Contact Support Log in to the Software Community for support of this and all of our connectors. The Software Community is your one-stop shop for self service and support.

Jira Connector Revision History

- 4.8.2-master -- March 2020
  - Defects fixed:
    - Added use of AccountID tag in JiraConnection to support *.atlassian.net (Jira OnDemand) target
- 4.8.1-master --- March 2019
  - Defects fixed:
    - Fixed one-to-many mapping for EnumFieldHandler
    - Fixed XML external references
- 4.8.0-master --- May 2018
  - Enhancements:
    - Unified distribution, requires use of Ruby 2.2.6 for TLS V1.2 support.
  - Changes:
    - Elimination of dependency on nokogiri requires change in the read_config method for all custom field handlers.
  - Fixes:
    - Support for Jira datetime fields.
- 4.7.5-master --- 30-Nov-2017
  - Fixes:
    - Fixed issue with range selector subset while using !=.
- 4.7.4-master-177 --- 12-Oct-2017
  - Fixes:
    - Updated code to follow the redirect URL whenever a 302 response is encountered for a GET request on an attachment. This resolves the issue of attachments not copying from On Demand Jira to AC.
- 4.7.3-master --- 23-Aug-2017
  - Fixes:
    - Modified construct_selector_syntax to add OR field is null condition when the original condition is field `!= <some_value>`.
    - Modified the construct_selector_syntax method to quantify the field as well as the value.
    - Replaced unassigned other_value with value in RallyEnumFieldHandler.transform_out with mapped value is nil.
- 4.7.2-master-174 --- 08-Nov-2016
  - Fixes:
    - Adjusted Time Zone conversion so Target Date field in maps correctly.
- 4.7.0--master--173 --- 23--Sep-2016
  - Enhancements:
    - Support for Jira 7
    - Subset and range copy and update selectors
    - Support for timetracking fields, specifically OriginalEstimate and RemainingEstimate.
- 4.6.5-master-167 --- 24-Nov-2015
  - Enhancements:
    - Detecting duplicate project names will now create an error log and exit condition.
    - Emailer functionality is now process prioritized to allow alert emails to send even if subsequent configuration validation fails.
    - Enhanced credential value encryption.
- 4.6.4-master-166 --- 16-Nov-2015

  - Enhancements:
    - Brand redesign.
- 4.6.2-master-165 --- 14-Aug-2015
  - Fixes:
    - Cleaned up log errors when the pre_copy method called on copy_to_rally can not satisfy the necessary conditions.
    - XML include files containing Url or Workspace or Project handled properly both before and after credential encryption.
- 4.6.1-master-158 --- 25-Jun-2015
  - Fixes:
    - Modified JiraConnection checkResultAgainstIntent method called by Connector.copy_to_other to handle Status field correctly.
    - Adjusted XML Schema Definition for Jira REST Connector to allow APIKey entry in RallyConnection section of the configuration file.
- 4.6.0-master-156 --- 31-May-2015
  - Enhancements:
    - Credentials in the RallyConnection and JiraConnection sections are now encrypted.
- 4.5.5-master-152 --- 25-April-2015
  - Enhancements:
    - Support for using APIKey in place of username and password.
    - Better handling of FieldDefaults and values.
    - Better logging of failed attempt to copy/update item from another system to .
- 4.5.4-master-149 --- 02-Apr-2015
  - Enhancements:
    - Better logging for create/update operations when error conditions encountered.
    - Better handling of string fields mapped to Name, Description, Notes that contain non-ASCII characters.
- 4.5.3-master-148 --- 07-Mar-2015
  - Enhancements:
    - Updated to use WSAPI v2.0.
    - Added a RallyBooleanFieldHandler.
    - Updated faraday gem to 0.9.0.
  - Fixes:
    - Fixed issue in YetiEmailer where header item had leading spaces not accepted by some email servers.
- 4.4.13-maintenance-92 --- 22-Jan-2015
  - Enhancements:
    - Support added for a RallyReferenceAttributeLookupFieldHandler intended to be used primarily with Release and Iteration fields where the attribute used for the lookup is other than the 'Name'.
    - This release is likely to be the last for the 4.4.x line that uses WSAPI 1.43.
- 4.4.12-maintenance-88 --- 02-Dec-2014
  - Fixes:
    - Corrects an issue with copying attachments to ALM.
- 4.4.11-maintenance-86 --- 18-Nov-2014
  - Fixes:

    - Fixed incorrect /Jira comment comparison problem from Jira 6.0 to 6.4 behavior change that caused comments to be copied multiple times.
    - Fixed defect of not recognizing standard Jira system date field (CreatedDate) in a CopySelector entry.
    - Fixed defect of retrieving only up to 50 users for each starting letter, now has a max of 1000 items per first letter of user name.
    - Fixed packaging defect for Linux/Mac builds where multiple versions per gem may have been included (although gemspec prevented use of non-specified versions).
- 4.4.10-master-140 --- 16-Oct-2014
  - Enhancements:
    - Modified rallyeif-wrk core dependency on rally_api to 1.1.2 (that depends on httpclient 2.4.0) to address SSLv3 security issue.
    - Added support for command line arguments to specify log file max size and file count.
    - Added configurable communication timeouts for Jira.
    - Added better formatting for Jira field schema output in debug mode.
- 4.4.4-master-132 --- 18-Sep-2014
  - Enhancements:
    - Improved semantic comparison of boolean values to strings ('true' / 'false') should result in fewer unnecessary updates
    - Improved semantic comparison of decimal values to integers/strings (1.0 should equal 1 or "1") should result in fewer unnecessary updates.
    - Improved comparison of User to user identifier from Jira should result in fewer unnecessary updates.
- 4.4.3-master-131 --- 29-Aug-2014
  - Enhancements:
    - Modifications to make parsing RevisionHistory Revision records more robust, especially the Notes field.
- 4.4.2-master-128 --- 07-Jul-2014
  - Enhancements/Fixes:
    - Common Reference fields (Project, Release, Iteration) no longer need to have a RallyReferenceFieldHandler configured in the Connector - RallyFieldHandlers section of the configuration file.
    - Mapping of Reference fields such as Release and Iteration is fully project scoped to ensure that if the DisplayName of multiple Release items is identical, the artifact contains the link to the item associated with the Project setting in the artifact.
- 4.3.0-master-89 --- 15-Jan-2014
  - Enhancements/Fixes:
    - Upgrade from Ruby 1.9.2-p290 to Ruby 2.0.0-p247
    - Performance improvement gained by using rally_api rather than rally_rest_api
    - Added more build tests
    - Added `<SuppressDeprecationWarning>` XML tag
    - Connector could not successfully map CreationDate to Due Date (DE18781)
    - Connector did not complain when it could not link a Bug to a Defect already copied to (DE17873)
    - Connector would validate only the first configuration file on command line (DE17492)
    - Upgrade to version 1.42 of the WSAPI
    - For the 4.3.0 version of this connector, any custom code (such as a custom field handler) will require modifications. A working example (using the QC connector) can be found here.

      Previous method for doing requires:

```
require 'yeti/field_handlers/other_field_handler'
require 'rexml/document'

class MyCustomFieldHandler < ....FieldHandler
```

```
    ....
end
```

New way of doing requires:

```
require 'rallyeif/wrk/field_handlers/field_handler'

module RallyEIF
  module WRK
        module FieldHandlers

          class MyCustomFieldHandler < ....FieldHandler
                 # See example code in installation directory:
                 # ./field_handlers/my_custom_field_handler.rb
           end
         end
    end
end
```

- 3.3.1-master-1012 --- 23-Aug-2013
  - Enhancements/Fixes:
    - Allow updating of Closed issues in Jira with Changed setting. Caveat: If the Jira Change is not made, the connector will crash out and stop the services
    - Updated version numbers
    - Rewrote version output method
    - Now uses Ruby's time for current time rather then the Jira server time
    - Version information is printed stdout (by -v) as well as rallylog.log
    - Fixed: Only first configuration file on command line was validated
    - Several small improvements in error messaging
- 3.2.0-master-1002 --- 31-May-2013
  - Enhancements/Fixes:
    - Updated rally_api gem to 0.9.14
    - Updated faraday gem to 0.8.7
    - Updated rally_jest.gem to 1.1.10
    - Added support for communicating with Jira through a Proxy with settings in the JiraConnection section of the configuration file
    - Changed Emailer sub-tag of Level to LogLevel
    - Dropped support for JiraHtmlFieldHandler for Description field
    - Versions 3.2.0 and greater are compatible with Jira 5.2 or greater.
- 3.0.1-48 --- 03-Apr-2013
  - Enhancements/Fixes:

    - Added install.sh to ease installation and ensure gem installation sequence
    - Check for appropriate version of Jira (must be 5.2 or better)
    - Rewrite of the existing Jira connector to use the Jira 5 REST API
    - Updated rally_api gem to 0.7.6
    - Introduced faraday gem 0.8.4
    - Dropped requirement of jira4r and soap4r gems
    - Field names for Jira specified in the configuration file must use Jira Display Name format
    - Dropped requirement to use `<JiraCollectionFieldHandler>` for Jira Affects Versions, Fix Versions and Components fields
    - Affects Versions, Fix Versions and Components fields in Jira can be bidirectionally mapped with fields
    - Jira connection section in configuration file is now called `<JiraRestConnection>` instead of `<JiraConnection>`
  - Known issues/limitations:
    - Versions 3.0.0 and greater are not compatible with Jira 4.x

Sample Configuration Files for Atlassian Jira

The following sample configuration files may help you with your integration between and Atlassian Jira.

- Jira-config-bugAttachment2Rally.pxml
- Jira-config-emailer-noauth.pxml
- Jira-config-Emailer.pxml
- Jira-config-RallyJiraCommentLinker.pxml
- Jira-config-TestConnection.pxml
- jira-config-us2task.pxml
- JIRA5rest-config-CrosslinkUrlField.pxml
- JIRA5rest-config-DueDate.pxml
- JIRA5rest-config-simple.pxml
- JIRA5rest-config-storyAttachment.pxml
- JIRA5rest-config-TestConnection.pxml
- JIRArest-Labels-to-Tags.pxml

For other useful setup information, see the Integrations Best Practices topic.

BROADCOM®
SOFTWARE