Rally Connector for XML – Carestream Branch

## History

## 4.0.0 - 22 Oct 2014 - Initial drop

## Introduction

The Rally Connector for XML provides synchronization capability between Rally and file systems via two services, described below. This version of Rally connectors is specifically designed for Carestream.

## Overview of Features

The Rally Connector for XML runs as a Ruby script on your server. The connector is configured via an XML file. A log file is created to track all actions made by the connector. The connector requires a custom field that exists in each system to store the unique id of the linked objects in the other system. The connector copies fields from Rally based on a field mapping that the user has specified in the configuration file.  Standard and custom fields can be mapped between the two systems.

The connector provides four services, but only two of them are activated at this point for synchronizing objects between Rally and the file system:

* Copy items (defects) created in Rally to the file system (COPY\_RALLY\_TO\_OTHER). The connector will create an XML file on the file system containing newly created Rally objects in standard format.
* Update items (defects) modified in Rally to the file system (UPDATE\_RALLY\_TO\_OTHER). The connector will create an XML file on the file system containing Rally objects that have an associated other ID and were updated since the last run.

The configuration file specifies which services to run, and in what order. At this time, the connector does not allow for the copying of XML data from the filesystem back into Rally. ***Remember that a separate configuration file is required for each record type.***

The output file will look like this:

<?xml version="1.0" encoding="UTF-8"?>

<Results>

<Defect>

<Description>blah blah blah</Description>

<Name>my defect</Name>

<Priority>None</Priority>

<Severity>Major Problem</Severity>

<State>Open</State>

<Duplicates>

<defect ref="https://server/slm/webservice/1.31/defect/1234" other\_i\_d="8675309" name="defect name" formatted\_i\_d="DE19"/>

<defect ref="https://server/slm/webservice/1.31/defect/1235" other\_i\_d="waiting XML" name="defect name" formatted\_i\_d="DE21"/>

</Duplicates>

</Defect>

<Defect>

...

</Defect>

</Results>

## Software/Hardware Requirements

The following are the software/hardware requirements:

1. A Rally subscription. Rally administrator privileges are needed for setup but not to run the connector.  You will need to know the URL for the Rally server hosting your subscription.  For example, rally1.rallydev.com.
2. The account running the connector needs to have the right to create files.
3. If you are using a proxy server, know the server and port of the proxy server.
4. Ruby 1.9.2 or greater (connector will not work with Ruby 1.8.x)

\*Note: To install Ruby on Windows, we recommend the "one-click installer" which can be found on [RubyInstaller.org](http://rubyinstaller.org/download.html).  You must install as an administrator on Windows for the path and environment variable settings to take effect.  By default, Ruby 1.9.2 will be installed in the C:\Ruby19 directory.

## Installation

### Basic Installation Steps

The are five steps to install. They are described in detail in the paragraphs below. The four steps are:

1. Install the Rally Connector code and Ruby gems. This assumes you have already installed Ruby on the machine where you will run the connector.
2. Make configuration changes in Rally and the other system.
3. Create a directory into which XML files will be written.
4. Edit configuration file to reflect your environment.
5. Run rally2\_xml\_connctor.rb to start the synchronization process.

#### Install the Rally Connector Code & Ruby Gems

1. (Optional) Set up the http\_proxy environment variable for Ruby if using a proxy server.  On Windows, right-click 'My Computer' and select 'Properties'. Select the 'Advanced' tab and then click 'Environment Variables'. Click 'New' under 'System Variables' and enter *http\_proxy* as the full HTTP URL, e.g.[http://www-cache:8000](http://www-cache:8000/).  
   \*Note that you will need to restart your system for Ruby to be able to see the environment variable.
2. Unzip and extract the contents of the zip file locally on your machine (e.g. C:\rally). The contents of the zip include these files:

* example.xml - An example configuration file.
* rally2\_xml\_connector.rb - Ruby file that you will use to run the connector.
* rallyeif-wrk.0.5.4.gem - library that includes the code needed to connect to Rally.
* rallyeif-xml-4.0.0.gem - library that includes the code needed to write the XML output.
* WinTail.exe - Utility to tail the log file output for debugging purposes.

Note: WinTail is a free text file tail utility for Windows, generously made available by Bare Metal Software Pty Ltd. for redistribution without restriction.  We have found it to be far superior to Notepad and other text editors for watching log files in real time as they are updated by the connector.

3. Install the provided gems by navigating to wherever you extracted the zip file:

gem install rallyeif\*.gem

Or when using a proxy:

gem install -p http://proxyhost:portnumber rallyeif\*.gem

#### Make Configuration Changes in Rally

*Note: See special section concerning test runs/test results below.*

**Create an External ID Field in Rally**

*Repeat for each record type you wish to sync.*

1. Log into Rally as a Workspace or Subscription administrator.
2. Navigate to Setup >> Workspaces & Projects.
3. Click on the appropriate Workspace that you wish to map to the other system. This will take you to the Details page for the given Workspace.
4. Click on Work Products & Fields and ensure the Work Product Type selected is *the appropriate one.*
5. Click Actions >> New Field.
6. Enter a 'Name' of ExternalID, 'Display Name' of ExternalID (name and display name must match), and type of 'String'.

Make note of the name of this field.  Once you start using the connector, this will contain the id of the work item as it exists in the other system.

#### Edit the Configuration File

An example configuration file is included in the delivered zip file and should be in the same directory where you extracted the Ruby gem.  For file systems, the default configuration file has a name of example.xml. We recommend making a backup copy of the example.xml in case you need to reference a valid configuration file later.

Edit the configuration file and enter the appropriate values between each begin and end tag.  ***You will need to make a separate configuration file for each record type that needs to be synchronized. The example below is for defects.***

<config>

  <RallyConnection>

    <Url>rally1.rallydev.com</Url>

    <WorkspaceName>Workspace Name</WorkspaceName>

    <Projects>

      <Project>Rally Project 1</Project>

<Project>Rally Project 2</Project>

    </Projects>

    <User>user@company.com</User>

    <Password>password</Password>

    <ArtifactType>Defect</ArtifactType>

    <ExternalIDField>ExternalID</ExternalIDField>

    <CreationFilter>ScheduleState = Open</CreationFilter>

  </RallyConnection>

  <XMLConnection>

<!-- user/password are required but ignored -->

    <User>None</User>

    <Password>None</Password>

<ArtifactType>Defect</ArtifactType>

<IDField>PLI\_ID</IDField>

    <ExternalIDField>RallyID</ExternalIDField>

<Path>/test\_dir/full\_cycle</Path>

  </XMLConnection>

  <Connector>

    <FieldMapping all\_rally\_fields=”true”>

      <Field><Rally>Name</Rally><Other>Title</Other></Field>

    </FieldMapping>

</Connector>

  <ConnectorRunner>

    <Preview>false</Preview>

<LogLevel>Debug</LogLevel>

    <Services>COPY\_RALLY\_TO\_XML, UPDATE\_RALLY\_TO\_XML</Services>

  </ConnectorRunner>

</config>

Each major section is surrounded by a tag to delineate that section. In the example above, here is a description of each section:

**RallyConnection** - Defines the connection information to Rally including Rally URL, username, password, etc.

**XMLConnection** -Defines the connection information for file system interaction including connection, artifact type, etc. For now, user and password are required but ignored. The user running the script must have permission to access and modify items on the file system.

**Connector -** Defines the field mapping between each system.  This expects each field is compatible with the field in the other system (Integer -> Integer, etc.)  The order of the fields in the mapping section determines the order in which those fields get updated.

**ConnectorRunner** –Specifies the services to run with this configuration. You can also set a Boolean to allow previewing of which records would get copied instead of the actual modifications.

#### Definition of Tags in Configuration File

Below is a more detailed explanation of each tag in the configuration file:

**Parent tag = <RallyConnection>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag Name** | **Description** | **Sample Values** | **Required?** |
| <Url> | Server used to connect to Rally. | Sandbox.rallydev.com  rally1.rallydev.com | Y |
| <WorkspaceName> | Workspace in Rally you want to copy/update work items. | My Workspace  YourWorkspace | Y |
| <User> | Login name for user to make the Web Services requests to create/update work items in Rally. | user@company.com | Y |
| <Password> | Password for user to make the Web Services requests to create/update work items in Rally.  *Note: The first time the connector runs, it will encode the password so it is not saved in plain text.* | Mypassword | Y |
| <ArtifactType> | Type of artifact you want to create/update in Rally. | Defect  HierarchicalRequirement  TestCase | Y |
| <ExternalIDField> | Rally 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 Rally' section above. | OtherId  ExternalID | Y |
| <Projects> | Contains a list of Project tags. Each tag refers to one Rally project that will be used when finding new Rally work items to copy to the other system.  For updating work items from Rally to the other system, all projects in <WorkspaceName> are considered.  At least one Rally project must be specified in this tag. | <Project>Rally1</Project> <Project>Rally2</Project> | Y |
| <CopySelectors>  <CopySelector>  string  </CopySelector>  </CopySelector> | Restrict the candidate stories to ones that make the match (name Rally fields when copying FROM Rally). When working with an iteration or release, you must use the ObjectID of the iteration or release and prepend it with "iteration/" or "release/", as appropriate. | ScheduleState = Open  Iteration = iteration/985223038 | N |

Tip: Escape special characters contained in a Rally Workspace or Project name that are markup sensitive. For example:

"&" ampersand becomes "&amp;"

">" greater than becomes "&gt;"

"<" less than becomes "&lt;"

**Parent tag = <XMLConnection>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag Name** | **Description** | **Sample Values** | **Required?** |
| <User> | Enter "NONE" | NONE | Y |
| <Password> | Enter "NONE" | NONE | Y |
| <ArtifactType> | Type of artifact you want to find/create/update. This will be used as the node name for each item. | User Story  Defect | Y |
| <Path> | Pathname to a directory (relative to your current working directory) in which to store the resulting XML files (one per run). The name will be formatted as ./<Path> /<number-of-seconds-since-Epoch>.xml. | /test\_dir/full\_cycle | Y |
| <ExternalIDField> | Custom string field that stores the unique id for the Rally system.  Refer to the 'Create an External ID Field' section above. | RallyID | N |

**Parent tag = <Connector>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag Name** | **Description** | **Sample Values** | **Required?** |
| <FieldMapping> | Determine what fields map between the two systems. The "Other" field name is what will be used as a node name for the output XML. | See 'Field Mapping' section below | **Y** |

**Parent tag = <ConnectorRunner>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag Name** | **Description** | **Sample Values** | **Required?** |
| <Preview> | Allows the user to enable a preview mode for testing where NO objects are copied/updated in either system. | false  true | N |
| <Services> | Determine what type of services to run.  Be careful if you update in both directions because which ever system runs the update first will overwrite the updates of the other system. Currently, only COPY\_RALLY\_TO\_XML and UPDATE\_RALLY\_TO\_XML are the only valid option. | COPY\_RALLY\_TO\_XML | Y |
| <LogLevel> | Determines what type of messages are written to the log file.  Highest level is 'Debug' (the default) where all messages are displayed. | Fatal, Error, Warn, Info, Debug(default) | N |

**Tip:** Incrementally set the connector up.  Start with a basic configuration file, test that you can connect to Rally in a "test" environment with one field.  Once you validate this is setup correctly, then start customizing the field mapping and field handler sections.

#### Field Mapping

The field mapping section is located between the <Connector> tags in the configuration file and defines what fields map between the two systems. In addition to mapping each Rally field name to an output field name, it's possible to ask for all of the Rally fields by adding the attribute "all\_rally\_fields" to the FieldMapping node. See the "All Fields" subsection below.

For example, this definition sets up a mapping between the Rally field 'Name' with the field 'Title' in the other system.  On a create and/or update, the connector will only update the Name field in Rally and the Title field in the other system (as well as the appropriate ID mapped fields).

<Connector>

  <FieldMapping all\_rally\_fields=”true”>

    <Field><Rally>Name</Rally><Other>Title</Other></Field>

  </FieldMapping>

</Connector>

When you setup 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/updated between the two systems and you will see an error in the log file.  For example, the connector will post an error for a particular work item if you try to post a string to a custom field of type 'integer' in Rally.

You can add subsequent mappings by appending to this list.  For example, this sets up a mapping to Rally Name >> Title, Rally Description >> Description, Rally Priority >> Importance:

<Connector>

  <FieldMapping>

    <Field><Rally>Name</Rally><Other>Title</Other></Field>

    <Field><Rally>Description</Rally><Other>Description</Other></Field>

    <Field><Rally>Priority</Rally><Other>Importance</Other></Field>

  </FieldMapping>

</Connector>

If you are mapping a drop-down value in Rally to the other system, this assumes that the drop-down values match. Otherwise, the connector will throw an error letting you know the value was not found in the list.

If your drop-down values are different between the two systems, see the next section called 'Mapping Drop-down Values'.

#### All Fields

It is no longer necessary to identify every field that you wish to extract from Rally. You can simply add the all\_rally\_fields attribute with a value of 'true' to the <FieldMapping> node. The following example will provide an output file with all the field nodes for the record type, where the output nodes have the same name as the field name inside Rally.

<Connector>

  <FieldMapping all\_rally\_fields='true'>

  </FieldMapping>

</Connector>

It is also possible to override a field name so that its output is a different name than what is in Rally. In the following example, the output file will have a node called "Description" (because it is one of the fields available and the user has not asked for its name to be changed) and a node called "Title", which will contain data from the Name field in Rally.

<Connector>

  <FieldMapping all\_rally\_fields='true'>

    <Field><Rally>Name</Rally><Other>Title</Other></Field>

  </FieldMapping>

</Connector>

Field Handlers (see below) can still be used as expected whether all\_rally\_fields is selected or not.

#### Mapping Drop-down Values (Optional)

If you have different drop-down values between the two systems, you can setup up another type of field handler in your configuration file with this information. The mapping section for drop-downs is located between the <Connector> tags in the configuration file and defines what field VALUES map between the two systems. For example:

<config>

...

<Connector>

   <OtherFieldHandlers>

     <OtherEnumFieldHandler>

       <FieldName>Importance</FieldName>

       <Mappings>

         <Field><Rally>Resolve Immediately</Rally><Other>1</Other></Field>

         <Field><Rally>High Attention</Rally> <Other>2</Other></Field>

         <Field><Rally>Normal</Rally> <Other>3</Other></Field>

         <Field><Rally>Low</Rally> <Other>4</Other></Field>

         <Field><Rally>Trivial</Rally> <Other>4</Other></Field>

         <Field><Rally>None</Rally> <Other>5</Other></Field>

       </Mappings>

     </OtherEnumFieldHandler>

   </OtherFieldHandlers>

</Connector>

...

</config>

This sets up a mapping of the values for the field Priority.  The value of the <FieldName> should be the name of the field in the other system.

The <Mappings> section defines the Rally value within the <Rally> tags and the corresponding values within the <Other> tags.  For example, 'Resolve Immediately' in Rally maps to a value of '1' in the other system and 'High Attention' maps to a value of '2' for the Priority field in the other system. Finally, a non-required drop-down in Rally has a 'None' value that you will want to map to a value in the other system.

You could add additional mappings to the configuration file by following this same format by adding another <OtherEnumFieldHandler> tags and then fill in the rest of the tags appropriately.

#### Mapping Users (Optional)

If you want to map users between Rally and the other system, you can specify an <OtherFieldHandlers><OtherUserFieldHandler> in the <Connector> section and include the <FieldName> and <Domain> tags. For example:

<Connector>

  <OtherFieldHandlers>

    <OtherUserFieldHandler>

      <FieldName>System.AssignedTo</FieldName>

      <Domain>company.com</Domain>

    </OtherUserFieldHandler>

  </OtherFieldHandlers>

</Connector>

The <FieldName> is the name of the field in the other system and the <Domain> determines the expected domain for usernames in Rally.  e.g. john in other system maps to john@company.com in Rally

In your mapping file, assuming you mapped the 'Owner' field in the other system to the Rally 'Owner' field:

<Connector>

  <FieldMapping>

    <Field><Rally>Owner</Rally><Other>System.AssignedTo</Other></Field>

  </FieldMapping>

</Connector>

Then the connector would map an 'Owner' value of 'john' to a Rally 'Owner' value of 'john@company.com'.

This assumes you have a naming convention between Rally and the other system where a user is just a *'name*' in the other system and a user in Rally is *'name****@domain****'*.

#### Mapping Project Fields (Optional)

If you want to map the project fields between Rally and the other system, first add a field mapping to the <Connector><FieldMapping> section:

<Field><Rally>Project</Rally><Other>Project</Other></Field>

### Run the Connector

Once the configuration file is set up, you can start running the connector.

ruby rally2\_xml\_connector.rb example.xml -1

to start the application. (The -1 will cause it to run once and end. Otherwise it will repeat its check every 15 minutes (or some other value if you put it in there. See the Multiple Configuration Files section below.)

If using Windows: Double-click on WinTail.exe to run WinTail.  In WinTail, open the file rallylog.log to watch the log messages generated by the connector.  This is an easy way to confirm that the connector is working properly, or to discover any errors.

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

Note for Windows Users: This software uses your Windows login credentials (not anything you put in the configuration file), so ensure that you are logged in as someone who has the right to modify the files. If you use the above method to cycle, then the system will have to remain logged in. We recommend creating a Windows Task for scheduling this work.

The script will create a timestamp file to identify the proper start point for the next run. This file will be named by inserting the word "time" into the configuration file name. In the example above, we would generate a time file that is named "r2x\_configtime.file."

#### Multiple Configuration Files

If you wish to map to more than one workspace in Rally, or map multiple artifact types and/or need to map to multiple targets, setting up multiple configuration files may make sense.

To run for different configuration files:

ruby rally2\_xml\_connector.rb config\_workspaceA.xml -1

ruby rally2\_xml\_connector.rb config\_workspaceB.xml -1

#### First Run Starting From Scratch

By default, if there is not an existing time file (that is, the script has not yet been run successfully), the script will start its search from five minutes before its execution. This only applies to the very first run, and the purpose is to prevent the unexpected copying of all the data ever collected in the tool. After a successful run, the time file now has the current time and the next run will search from that point. One easy way to force the script to grab earlier data is to modify the time file by hand.

It is also possible to execute the script with a switch to force this behavior: Use either -s or -scratch to indicate that the script should start from scratch and gather all the data.

## Troubleshooting the Connector

Once the connector is running, all errors are written to a log file in the directory where the rally2\_xml\_connector.rb script resides. Informational messages, warning and errors are written to the log file depending on the value of the <LogLevel> tag in the <ConnectorRunner> section of the config file.

To see the most recent log messages on Windows, rename the extension of the WinTail file from .dat to .exe and then double-click on the WinTail.exe and browse to the rallylog.log file.

Before the connector starts synchronizing objects between the two systems, it performs a validation that confirms that 1) Connecting to Rally is successful, 2) Connecting to the other system is successful, 3) Fields in the field mapping exist in Rally and the other system, and 4) Ensures that 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, ensure you set the <Preview> tag in the configuration file to 'true'. This acts as a good trick if you want to experiment with some different configuration options to debug an issue.

## 6.1 FAQ

1. *When the Defect is created in Rally and then pushed into the delivery XML file, it does not yet know what the other system's identifier will be, so the script pushes "waiting XML" into the external identifier field in Rally. The other system will need to tell Rally what its ID is.*
2. *Note that only fields that are populated will provide a node in the exported XML.*