Redmine to JIRA Importers plugin Documentation

Release 0.10.0

Michele Cardone

Jun 13, 2021

Contents

Contents:

CHAPTER 1

Redmine to JIRA Importers plugin

Export Redmine issues to file formats compatible with the JIRA Importers plugin (JIM).

- Free software: MIT license
- Documentation: https://redmine2jira.readthedocs.io.

1.1 Features

The aim of the tool is to export Redmine issues, fetched using Redmine REST API, to a set of files which format is compatible with the JIRA Importers Plugin.

The output of the tool, in most of the scenarios, is a single JSON file combining all the following information for each exported issue:

- Standard/custom fields
- Journal entries (Notes)
- Status history
- Attachments URLs
- Hierarchy relationships
- Relations
- Watchers
- Time logs

1.1.1 Cross-project issue relations

If the Redmine instance has configured cross-project issue relations, and the exported issues do not correspond to the full set of issues of the Redmine instance (the tool will properly detect the scenario and prompt a question if needed), the issue relations will be exported in a separate CSV file. Subsequently, when all the Redmine issues have been imported in the target Jira instance that CSV file can be finally imported in order to update relations on all the existing issues.

1.1.2 JIM file format specifications

Both the JSON and CSV files produced respectively meet their format specifications for the JIRA Importers plugin (JIM). Those specifications can be respectively found in the following KB articles:

- Cloud / Importing data from JSON
- Cloud / Importing data from CSV
- Server (latest) / Importing data from JSON
- Server (latest) / Importing data from CSV

However, it's worth to mention that all the articles, especially the one Related to JSON format, are more driven by examples rather than being comprehensive specification documents: several details related both to the structure and the fields values format are omitted. Sometimes we had the need to rely on other sources on the Internet to cope some strange scenarios. Besides, the import from JSON feature is not completely stable.

1.2 Prerequisites

- TODO Users already present in Jira
- TODO Redmine REST API Enabled

1.3 Usage

The '–filter' option accept a HTTP GET parameter string. Here follows the list of the supported filter parameters:

- issue_id (int or string): Single issue ID or comma-separated issue ID's
- project_id (int or string): Project ID/identifier
- subproject_id (int or string): Subproject ID/identifier (To be used in conjunction with 'project_id'; you can use *project_id=X* and *subproject_id=!** to get only the issues of a given project and none of its subprojects)
- tracker_id (int): Tracker ID
- query_id (int): Query ID
- status_id (int): ['open', 'closed', '*', id] If the filter is not specified the default value will be 'open'.
- assigned_to_id (int):_Assignee user ID (or 'me' to get issues which are assigned to the user whose credentials were used to access the Redmine REST API)
- cf_x: Custom field having ID 'x'. The '~' sign can be used before the value to find issues containing a string in a custom field.

NB: operators containing ">", "<" or "=" should be hex-encoded so they're parsed correctly. Most evolved API clients will do that for you by default, but for the sake of clarity the following examples have been written with no such magic feature in mind.

To fetch issues for a date range (uncrypted filter is "><2012-03-01|2012-03-07") : GET /issues.xml?created_on=%3E%3C2012-03-01|2012-03-07

To fetch issues created after a certain date (uncrypted filter is ">=2012-03-01") : GET /issues.xml?created_on=%3E%3D2012-03-01

Or before a certain date (uncrypted filter is "<= 2012-03-07") : GET /issues.xml?created_on=%3C%3D2012-03-07

To fetch issues created after a certain timestamp (uncrypted filter is ">=2014-01-02T08:12:32Z") : GET /issues.xml?created_on=%3E%3D2014-01-02T08:12:32Z

To fetch issues updated after a certain timestamp (uncrypted filter is ">=2014-01-02T08:12:32Z") : GET /issues.xml?updated_on=%3E%3D2014-01-02T08:12:32Z

1.4 Configuration

- TODO

1.5 Versioning

We use SemVer for versioning.

1.6 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

[~ Dependencies scanned by PyUp.io ~]

CHAPTER 2

Installation

2.1 Stable release

To install Redmine to JIRA Importers plugin, run this command in your terminal:

```
$ pip install redmine2jira
```

This is the preferred method to install Redmine to JIRA Importers plugin, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Redmine to JIRA Importers plugin can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/wandering-tales/redmine2jira
```

Or download the tarball:

```
$ curl  -OL https://github.com/wandering-tales/redmine2jira/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Redmine to JIRA Importers plugin in a project:

```
import redmine2jira
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/wandering-tales/redmine2jira/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Redmine to JIRA Importers plugin could always use more documentation, whether as part of the official Redmine to JIRA Importers plugin docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/wandering-tales/redmine2jira/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *redmine2jira* for local development.

1. Fork the *redmine2jira* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/redmine2jira.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv redmine2jira
   $ cd redmine2jira/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 redmine2jira tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.5 and 3.6, and for PyPy. Make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_redmine2jira
```

CHAPTER 5

Credits

5.1 Development Lead

- Michele Cardone <michele.cardone82@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.10.0 (2018-03-14)

6.1.1 New features

- Implemented issue journal details export feature
- Implemented issue category list by project feature
- Implemented version list by project feature

6.1.2 Improvements

- Added support to version resource type mappings
- Re-engineered definitions of internal domain entities and their mappings via classes and named tuples
- Moved definitions of internal domain entities and their mappings to ‘resources’ sub-package
- Refactored issues export feature to ‘IssueExporter’ class
- Improved and optimized description of resource mappings settings
- Slightly improved configuration settings comments
- Updated Sphinx to 1.7.1
- Several code optimizations

6.1.3 Changes

6.1.4 Fixes

- Added Python 2 Unicode compatibility for string type

- Used project identifier instead of its internal ID to fetch per-project static resource value mappings
- Used lists instead of sets to achieve Json format serializer compatibility
- Used safer method to check for journal notes existence before fetching them

6.2 0.9.0 (2018-02-11)

6.2.1 Improvements

- Update coverage from 4.5 to 4.5.1

6.2.2 Changes

- Disable possibility to skip dynamic value mapping feature
- Remove printing of issues referenced users at the end of export phase.

  As both static and dynamic value mappings are enabled for user resources, the final user doesn't need to be warned for something he consciously did in either case.

6.2.3 Fixes

- Honor value mappings for user resources

6.3 0.8.0 (2018-02-10)

6.3.1 New features

- Implemented issue watchers save method
- Implemented issue attachments save method
- Partially implemented issue journals save method. Redmine journal notes are saved to Jira comments.

6.3.2 Fixes

- Apply conversion to Confluence Wiki notation only if Textile or Markdown text formatting is enabled in settings

6.4 0.7.0 (2018-02-10)

6.4.1 New features

- Implemented issue custom fields save method

6.4.2 Improvements

- Used tuples as dictionary keys for both resource type fields mappings and dynamic resource value mappings
- Added comment to explain what happens when no static user-defined mapping has been found and dynamic resource value mapping feature is disabled
- Removed leftovers of old project name "Redmine XLS Export to Jira"

6.4.3 Changes

- Removed *CUSTOM_* prefix from the resource value mappings setting names

6.4.4 Fixes

- Fixed setting of dynamic resource value mapping when the resource type is defined on a per-project basis
- Removed misleading comment. When a Redmine resource type is mapped to more than one Jira resource type the final user is free to set value mappings across all possible resource type combinations.
- Added default empty dictionary if the resource type mapping setting is not found
- Minor docstring fixes

6.5 0.6.2 (2018-02-08)

6.5.1 Improvements

- Add pyenv support for Tox

6.5.2 Fixes

- Fix packages include directive in `setup.py`

6.6 0.6.1 (2018-02-07)

Fake release to fix a problem in PyPI upload.

6.7 0.6.0 (2018-02-07)

6.7.1 New features

- Implemented issue project save method
- Implemented issue standard fields save methods

6.7.2 Improvements

- Renamed `_get_resource_value_mapping` method to `_get_resource_mapping`.

  The method now returns both mapped Jira type and value, rather than only value.

  Updated method docstring accordingly.
- Added Redmine general configuration section header

6.7.3 Changes

- Removed Python 3.3 compatibility
- Updated encrypted PyPI password for Travis CI

6.7.4 Fixes

- Replaced references to old `CUSTOM_USERS_MAPPINGS` setting with new `CUSTOM_REDMINE_USER_JIRA_USER_MAPPINGS`
- Retrieved issue user resource instance from cached users list rather than from issue lazy loaded instance
- Disabled dynamic value mapping feature for Redmine "User" resource type

6.8 0.5.0 (2018-02-06)

6.8.1 New features

- Added dynamic resource value mapping management at runtime
- Added dynamic resource value mapping for assignee field when it refers to a standard user
- Added command to list issue priorities

6.8.2 Improvements

- Made Redmine and Jira respective resource types explicit in the names of settings related to resource value mappings
- Slightly improved settings related comments
- Added labels for values printed in console output
- Improved code readability
- Slightly improved docstrings
- Updated `sphinx` to 1.6.7
- Updated `coverage` to 4.5

6.9 0.4.0 (2018-01-26)

6.9.1 New features

- Added dynamic project mappings management

6.9.2 Improvements

- Refactored specific methods to save issue resources
- Minor optimizations

6.10 0.3.1 (2018-01-26)

6.10.1 Improvements

- Referenced users and groups are collected on-the-fly while exporting issues. This increases performance.
- Minor enhancements in the console output for the completion of the export

6.10.2 Fixes

- Fix recursive function used in `list projects` command to build the full project hierarchical name
- Fixed a bug affecting all the `list` commands that caused some resource relations being included in the tables
- Fixed another minor bug affecting all the `list` commands

6.11 0.3.0 (2018-01-22)

6.11.1 Improvements

- Added early lookup of users and groups references within the issues being exported
- Added command to list Redmine groups
- Added option to list all Redmine users at once, including locked ones
- Enhanced notes in configuration file

6.11.2 Changes

- Added requirements.txt for installation package requirements (useful for pyup.io)

6.12 0.2.0 (2018-01-19)

6.12.1 Improvements

- Added PyCharm IDE configuration and Python Virtual Environments to .gitignore
- Added configuration file with defaults and support for local configuration file
- Minor documentation fixes

6.12.2 Changes

- Dropped out “Redmine XLS Plugin” in favor of Redmine REST API.

  Since the files exported by the plugin lack some information needed to produce files compatible with the Jira Importer Plugin (JIM), several calls to the Redmine REST API were needed to compensate the data. Hence to avoid the effort to merge the data coming from two difference sources I decided to rely solely on Redmine REST API to fetch all the needed data.

  This is a major project scope change that implied, in turn, the following modifications:

  - Renamed GitHub repository from “redmine-xls-export2jira” to “redmine2jira”
  - Renamed Python package from “redmine_xls_export2jira” to “redmine2jira”
  - Rename project description to “Redmine to JIRA Importers plugin”

  Any other reference to the “Redmine XLS Export” plugin has also been removed from the documentation.

- Removed Python 2.7 compatibility. Added Python 3.6 compatibility.
- Temporarily disable CLI tests

6.13 0.1.1 (2018-01-05)

6.13.1 Fixes

- Minor fixes in docs

6.13.2 Improvements

- Initial pyup.io update
- Added pyup.io Python 3 badge

6.13.3 Changes

- Linked pyup.io
- Removed CHANGELOG.rst

6.14 0.1.0 (2018-01-05)

- First release on PyPI.

CHAPTER 7

Appendixes

7.1 Jira REST API Handbook

Here follows a list of Jira REST API's that you might find useful while working with the `redmine2jira` tool. For instance you may want to retrieve those nasty Jira internal resource ID's used in the dictionary mappings in the configuration files.

This handbook is organized in sections, each corresponding to a Jira resource type.

Note: Unless noted otherwise the internal ID field is always `id`. Should `id` not be included in the JSON response, the `key` field can be used in its place.

Note: Some API's return paginated list of resources, each specifying a maximum number of fetched resources per single call. In such cases you may want to call them multiple times properly tweaking the `startAt` and `maxResults` parameters.

In the URL examples below the `maxResults` parameter is set by default to the maximum number of results each API is able to return per single call.

7.1.1 Users

- Get a user, either active or inactive, by his username:

```
/rest/api/2/user?username={username}&includeInactive=True
```

- Get all users, both active and inactive:

```
/rest/api/2/user/search?startAt=0&maxResults=1000&username=_&includeInactive=True'
```

This API returns a paginated list of users. The maximum number of returned users is limited to 1000 (already set in the URL above).

- Get all users from group, both active and inactive:

```
/rest/api/2/group/member?groupname={group_name}&startAt={index}&maxResults=50&
↪includeInactiveUsers=True
```

This API returns a paginated list of users. The maximum number of returned users is limited to 50 (already set in the URL above). This API is useful if you configured a special group containing all of your users.

Note: The value of `key` field, for the current version of the Jira REST API (v2), is equal to the value of the `username` field, hence you may simply avoid calling the API's above if you need to retrieve user ID's.

7.1.2 Projects

- Get all projects:

```
/rest/api/2/project
```

- Get project:

```
/rest/api/2/project/{projectIdOrKey}
```

7.1.3 Issue Types

- Get all issue types:

```
/rest/api/2/issuetype
```

7.1.4 Issue Statuses

- Get all issue statuses:

```
/rest/api/2/status
```

- Get an issue status:

```
/rest/api/2/status/{idOrName}
```

7.1.5 Issue Priorities

- Get all issue priorities:

```
/rest/api/2/priority
```

7.1.6 Custom Fields

- Get all issue fields, both System and Custom:

```
/rest/api/2/field
```

7.1.7 Components

- Get project components:

```
/rest/api/2/project/{projectIdOrKey}/components
```

7.1.8 Labels

Unfortunately currently there are no endpoints in Jira REST API v2 to work with labels, and there is no way to workaround.

See https://jira.atlassian.com/browse/JRASERVER-29409

CHAPTER 8

Indices and tables

- genindex
- modindex
- search