kgish / **assembla-to-jira-migration**

A comprehensive toolset for migrating data from Assembla to Jira.

MIT License

1 star **1** fork

Unstar Unwatch

Code Issues 20 Pull requests 1 Actions Projects Wiki Security Insights

master

kgish ... 1 hour ago

View code

README.md

Assembla to Jira Migration

A collection of advanced tooling which provides seamless data migration from Assembla to Jira.

This is by far the best Assembla to Jira migration toolset around. Here are some of the reasons why this toolset beats out all others hands down:

- Fully automated requiring minimal manual actions.
- Configuration file with various options.
- Import users (names, emails, roles and permissions).
- Import tickets, comments, attachments and relationships.
- Retain ticket status, labels, ranking and custom fields.
- Link back to original Assembla tickets for reference.
- Save relevant Assembla context in user-defined fields.
- Embed image thumbnails in descriptions and comments.

- Convert markdown, users, urls, attachments and other links.
- Timetracking: estimated and remaining.
- Retain watchers of tickets for notifications.
- Create scrum or kanban board with workflow.
- Map the Assembla milestones to Jira sprints.
- Populate the backlog, future and current sprints.
- Assign stories to epics.
- Resolve cross linking between external projects.
- Export wiki to Confluence space (new!)
- Account for the API differences between hosted and cloud.
- Tons of great documentation and trouble-shooting guide.

If you need extra help just look here.

Introduction

Have you ever wanted to use Jira instead of Assembla, but were afraid that the switch to Jira was too risky?

Are you worried that your business-critical data in Assembla will get corrupted or even lost during the conversion?

Jira already offers a number of standard add-ons to make certain migrations easier. Unfortunately, it does not offer a tool for migrating from Assembla. In fact, they have announced that they will NOT provide a native JIRA import from Assembla in the foreseeable future, bummer.

However, **you are now in luck!** By using this Assembla-to-Jira migration toolset, it is very easy to export all of the relevant Assembla data and import most (if not all) of it into a Jira project without loss or corruption of data.

By using the Assembla API and the Jira API together, both environments are hooked up in order to make all necessary data transformations run smoothly and automatically.

Most of the actions are done automatically via a pipeline of scripts. Just define the required parameters in the `.env` configuration file, and you are ready to go.

Some manual actions are required since the Jira API does not support all of the required data transformations, however these actions are few and clearly documented below. It is very important NOT to skip these manual actions because a successful migration depends on them being done properly at the right time.

While not required, it is still best to start with a fresh installation, e.g. one in which the desired project has not yet been created and none of the project users are present yet. Otherwise, unexpected problems might occur.

At the beginning of the migration, a quick scan is made to detect if a given project already exists. If not then the new project is created for you, as well as all of the Assembla users who have not yet been created.

Although the official Atlassian documentation states that the Jira API for hosted server is nearly identical to the cloud server, there are some subtle, tricky differences that can bite you when you least expect.

Don't worry, if you read all of the instructions below, follow them carefully without skipping anything, you can avoid these annoying bumps in the road on your way to a successful migration.

Need help? Look here.

Installation

The toolset has been written with the Ruby programming language. In order to be able to use it, you will have to have downloaded and installed the following items on your computer:

- Ruby
- Bundler
- Git

Ensure that you have the correct version of ruby installed and set to using it for scripts.

```
$ rvm install `cat .ruby-version`
$ rvm use `cat .ruby-version`
```

Once this has been done, you can checkout and install the toolset from the github repository.

```
$ git clone https://github.com/kgish/assembla-to-jira-migration.git
assembla-to-jira
$ cd assembla-to-jira
$ gem install bundler
$ bundle install
```

At this point everything should be ready to use as explained in the following sections.

Supported API version

Please note that only the following API versions are supported:

- Assembla API v1 `api.assembla.com/v1`
- JIRA API v2 `rest/api/2`

Pipeline Steps

The complete migration from start to finish consists of a series of scripts which are to be executed in order.

Like a pipeline, each script processes data and generates a dump file to store the intermediate results. This output is used in turn as input for the following script.

The reason for doing this that if something goes wrong you do not lose everything and can restart from the previous step.

Each step will generate a log of the results in the form of a csv file for reference purposes, e.g. detecting which requests failed and why. For example, importing tickets will create the `data/jira/:space/jira-tickets.csv` file where `:space` is the Assembla space name `ASSEMBLA_SPACE` in the `.env` file hyphenated with the `hostname-port`. For example: `space-name-jira-example-org-8080`.

While the script is being executed, information will be logged to the console. Be sure to inspect the information, as certain instruction might be given that you must follow before continuing to the next step.

Assembla export

First we export all the data from Assembla.

1. Space (spaces, space_tools, users, user roles, tags, milestones, ticket statuses, ticket custom fields, documents, wiki pages and tickets)
2. Tickets (comments, attachments, tags, associations)
3. Report users
4. Report tickets

Jira import

Now that all of the Assembla data is available, we can now take this and import it into Jira.

5. Create project (and board)
6. Create issue link types
7. Get settings (issue types, priorities, resolutions, roles, statuses and projects)
8. Import users

9. Download ticket attachments
10. Create custom fields
11. Import custom fields
12. Import tickets
13. Resolve/update ticket links
14. Import ticket comments
15. Import ticket attachments
16. Update attachment links
17. Update ticket status (resolutions)
18. Update ticket associations
19. Update ticket watchers
20. Resolve/update ticket and comment external links
21. Move stories to epics
22. Move stories to epics which could not be exported
23. Rank tickets (cloud only)

Scrum/Kanban board

Using the Agile extension, create the sprints and populate the scrum/kanban board.

24. Create sprints
25. Update board

Wiki to Confluence migration

26. Migrate Assembla wiki to Confluence space.

Cleanup

27. Manual cleanup actions

Congratulations, you did it!

Preparations

You will need to go to to the Jira hosted (`jira.example.org`) or cloud (`id.atlassian.net`) instance and login as admin.

Create the admin user as defined in the `.env` file as `JIRA_API_ADMIN_USERNAME` and ensure that this user is activated and belongs to the following groups:

- site-admins

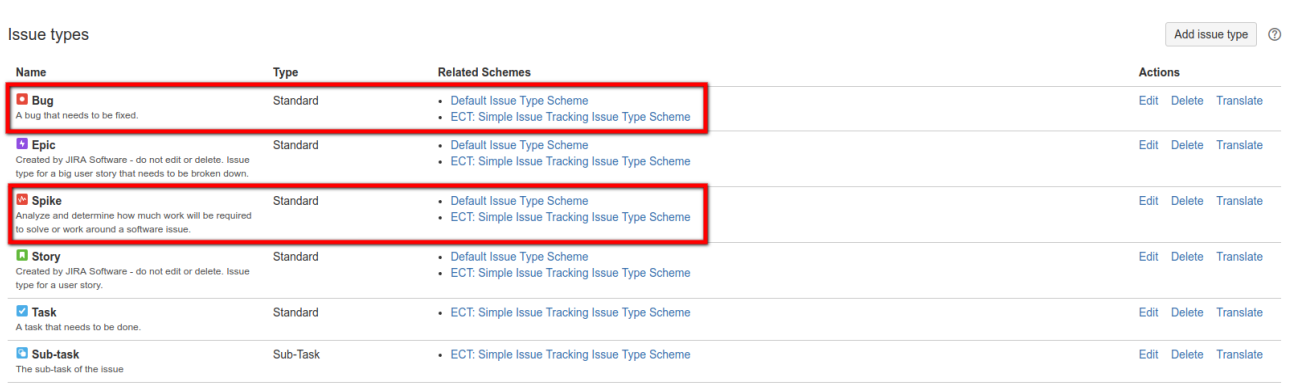
- jira-administrators

Create the following new issue types:

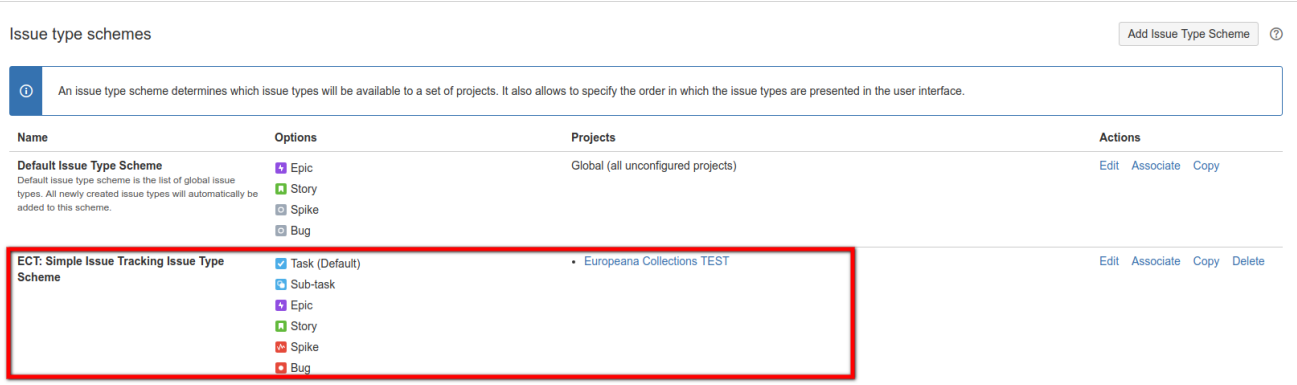
- Spike
- Bug (if not already present)

Those tickets whose summary starts with `Spike:` or `Bug:` will acquire these issue types so it is important that they already exist.

These are defined in the `.env` file, see `ASSEMBLA_TYPES_EXTRA` below.



You will also need to configure the `issue type scheme` for the project like this:



Environment

An example configuration file `.env.example` is provided for you to define a number environment parameters which affect the behavior.

```
# --- General settings --- #
DATA_DIR=data
TICKETS_CREATED_ON=YYYY-MM-DD
DEBUG=false

# --- Assembla settings --- #
ASSEMBLA_API_HOST=https://api.assembla.com/v1
ASSEMBLA_API_KEY=api-key
ASSEMBLA_API_SECRET=api-secret
ASSEMBLA_URL_TICKETS=https://app.assembla.com/spaces/[:space-name]/tickets
```

```
ASSEMBLA_SPACE=[:space-name]
ASSEMBLA_WIKI=https://[:company-name].assembla.com/spaces/[:space-
name]/wiki
ASSEMBLA_WIKI_NAME=[:space-name]
ASSEMBLA_SKIP_ASSOCIATIONS=parent,child,story,subtask
# Ticket types extracted from ticket summary, e.g. starting with 'Spike: '
ASSEMBLA_TYPES_EXTRA=spike,bug
ASSEMBLA_CUSTOM_FIELD=field-name

# --- Jira API settings --- #/
# Server type must be 'hosted' or 'cloud'
JIRA_SERVER_TYPE=cloud
# Base must start with 'https?://'
JIRA_API_BASE=https://jira.example.org
JIRA_API_HOST=rest/api/2
JIRA_API_PROJECT_NAME=name
JIRA_API_PROJECT_KEY=key
JIRA_API_KEY=secret
# Project type must be scrum (default) or kanban
JIRA_API_PROJECT_TYPE=scrum
JIRA_API_ADMIN_USER=john.doe
JIRA_API_ADMIN_PASSWORD=secret
JIRA_API_ADMIN_EMAIL=john.doe@example.org
JIRA_API_UNKNOWN_USER=unknown.user
JIRA_API_DEFAULT_EMAIL=example.org
JIRA_API_IMAGES_THUMBNAIL=description:false,comments:true
JIRA_API_SKIP_EMPTY_COMMENTS=true
JIRA_API_SKIP_COMMIT_COMMENTS=true

JIRA_API_STATUSES=New:To Do,In Progress,Code Review:Review,Test,Test: In
Progress:Test,Ready for Deploy:Ready,Re-opened:To
Do,Fixed/Closed:Done,Deferred:To Do,Invalid/Duplicate:Done

# Cross project ticket linking
JIRA_API_SPACE_TO_PROJECT=space1-name:project1-key,space2-name:project2-
name
JIRA_API_RE_TICKET=https?://.*?
\.assembla\.com/spaces/(.*?)/tickets/(\d+)...
JIRA_API_RE_COMMENT=https?://.*?
\.assembla\.com/spaces/(.*?)/tickets/(\d+)...
JIRA_API_BROWSE_ISSUE=browse/[:jira-ticket-key]
JIRA_API_BROWSE_COMMENT=browse/[:jira-ticket-key]?focusedCommentId=...

# --- Jira Agile settings --- #
JIRA_AGILE_HOST=rest/agile/1.0
```

By using the filter `TICKETS_CREATED_ON` you can limited the tickets to those that were created on or after the date indicated. So for example:

```
TICKETS_CREATED_ON=2017-06-01
```

would only include those tickets created on or after the first of June in the year 2017.

IMPORTANT: Using this settings will result in some ticket links that cannot be resolved any more, since they were created in the past and not included in the import data.

```
$ cp .env.example .env
```

Jira hosted versus cloud

Although the official Jira documentation claims that the hosted and cloud APIs are identical, I've found out that this isn't entirely true. There are a couple of minor differences that must be taken into account:

- Performance - The cloud server is MUCH slower than the hosted server, meaning that when imported long lists of tickets or whatever extreme patience is required.
- Stability - The hosted server generally works flawlessy and completes after the first run whereas the cloud server occasionally times out or returns a server error. For example, importing attachments.
- Users - When creating users the hosted server will automatically set activated to true, whereas the cloud server will NOT. In addition, the server can take too long causing the connection to timeout, meaning that you have to keep re-running the script until all of the user are created.
- Reporter - The hosted server will allow you to set the reporter when creating issues while the cloud server will NOT.
- Ranking - The hosted server will allow you to set the issue rank when creating issues while the cloud server will NOT.
- Comments - The hosted server will allow original comment authors to import comments while cloud server will NOT.
- Attachments - The cloud server is problematic, and certain extra actions must be taken.
- Storage - The cloud server imposes a file upload size limit (100MB). If possible, compress the file and try again my manually adding it as an attachment to the Jira issue.

The Jira server type is given by `JIRA_SERVER_TYPE` which is defined as either `hosted` or `cloud`, and is detected automatically by the call:

```
GET /rest/api/2/serverInfo
def jira_get_server_type
  ...
end
```

where the value of `response['deploymentType']` is used: `Server => hosted` or `Cloud => cloud`. This value is cached in the `jira-serverinfo.csv` dump file.

Make sure you're using your Atlassian account email address and password for basic authentication, not your Jira username.

Export data from Assembla

You can run the export in a number of stages, output files being generated at each point in the process.

Make sure that you are using the correct version of ruby.

```
$ rvm use `cat .ruby-version`
```

The output files are located in the directory `data/assembla/:space/` as follows:

```
$ ruby 01-assembla_export_space.rb # => space-tools.csv, users.csv, user-
roles.csv, ticket-tags.csv, \
    milestones-all.csv, tickets-statuses.csv, tickets-custom-fields.csv,
documents.csv, \
    wiki-pages.csv, tickets.csv
$ ruby 02-assembla_export_tickets.rb [type] # => ticket-comments.csv,
ticket-attachments.csv, \
    ticket-tags.csv, ticket-associations.csv
$ ruby 03-assembla_report_users.rb # => report-users.csv
$ ruby 04-assembla_report_tickets.rb # => report-tickets.csv
```

Executing `01-assembla_export_space.rb` can be very time consuming, so you might want to break it up into smaller chunks and running them in parallel, by passing the optional `type` (space_tools, users, user_roles, tags, milestones/all, tickets/statuses, tickets/custom_fields, documents, wiki_pages, tickets) as the first argument.

```
$ ruby 01-assembla_export_space.rb space_tools # => space-tools.csv
$ ruby 01-assembla_export_space.rb users # => users.csv
$ ruby 01-assembla_export_space.rb user_roles # => user-roles.csv
$ ruby 01-assembla_export_space.rb tags # => ticket-tags.csv
$ ruby 01-assembla_export_space.rb milestones/all # => milestones-all.csv
$ ruby 01-assembla_export_space.rb tickets/statuses # => tickets-
statuses.csv
$ ruby 01-assembla_export_space.rb tickets/custom_fields # => tickets-
custom-fields.csv
$ ruby 01-assembla_export_space.rb documents # => documents.csv
$ ruby 01-assembla_export_space.rb wiki_pages # => wiki-pages.csv
$ ruby 01-assembla_export_space.rb tickets # => tickets.csv
```

The same applies when executing `02-assembla_export_tickets.rb` , and you can break it up into smaller chunks by passing the optional `type` (comments, attachments, tags or associations) as the first argument.

```
$ ruby 02-assembla_export_tickets.rb comments # => ticket-comments.csv
$ ruby 02-assembla_export_tickets.rb attachments # => ticket-
attachments.csv
$ ruby 02-assembla_export_tickets.rb tags # => ticket-tags.csv
$ ruby 02-assembla_export_tickets.rb associations # => ticket-
associations.csv
```

This allows you to recover better to the previous step in case of failure, for example near the end where you would lose all the data in the dump files.

If you also want to capture the output, then you can run the above commands like this:

```
$ ruby nn-assembla_xxx.rb | tee logs/nn-assembla_xxx.log
```

In other words:

```
$ ruby 01-assembla_export_space.rb | tee logs/01-assembla_export_space.log
$ ruby 02-assembla_export_tickets.rb | tee logs/02-
assembla_export_tickets.log
$ ruby 03-assembla_report_users.rb | tee logs/03-assembla_report_users.log
$ ruby 04-assembla_report_tickets.rb | tee logs/04-
assembla_report_tickets.log
```

And watch the progress as follows:

```
$ tail -f logs/nn-assembal_xxx.log
```

Import data into Jira

You can run the import in a number of stages, output files being generated at each point in the process.

Create project (and board)

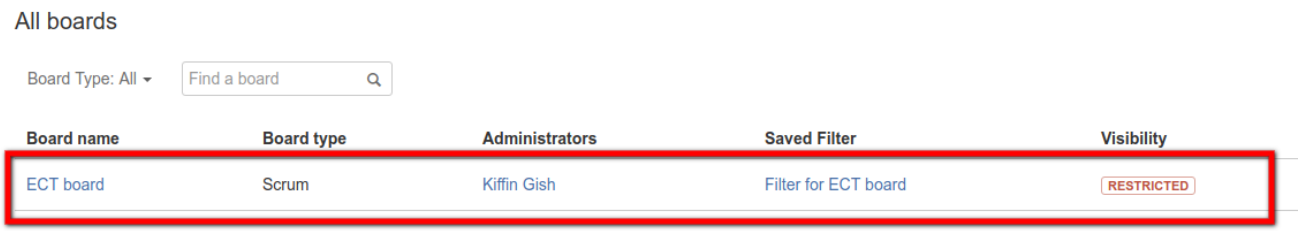
```
POST /rest/api/2/project
{
  key: project_key,
  name: project_name,
  projectTypeKey: 'software',
  description: project_description,
  projectTemplateKey: "com.pyxis.greenhopper.jira:gh-#{type}-template",
  lead: username
}
```

where `#{type}` must be either `scrum` or `kanban`.

```
$ ruby 05-jira_create_project.rb # => data/jira/:space/jira-serverinfo.csv
```

Depending on the value of `JIRA_API_PROJECT_TYPE` in the `.env` file, a scrum or kanban board will be created as well with board name `{projectKey} board`.

The `projectKey` is usually just the abbreviation of the project name in all capitals. Here is an example of a project with key `ECT`:



Create issue link types

```
POST /rest/api/2/issueLinkType
{
  name: name,
  inward: inward,
  outward: outward
}
```

Execute the followng command:

```
$ ruby 06-jira_create_issuelink_types.rb # => data/jira/:space/jira-
issuelink-types.csv
```

Get general information

Some extra general information needs gathering before the migration can start.

```
GET /rest/api/2/{issuetype|priority|resolution|role|status|project}
```

Execute the following commands:

```
$ ruby 07-jira_get_info.rb # => data/jira/:space/jira-issue-types.csv
```

which will generate the following output file in the `data/jira/:space` directory:

- jira-issue-types.csv
- jira-priorities.csv
- jira-resolutions.csv
- jira-roles.csv
- jira-statuses.csv
- jira-projects.csv

Import users

```
POST /rest/api/2/user
{
  name: user['login'],
  password: user['login'],
  emailAddress: user['email'],
  displayName: user['name']
}
```

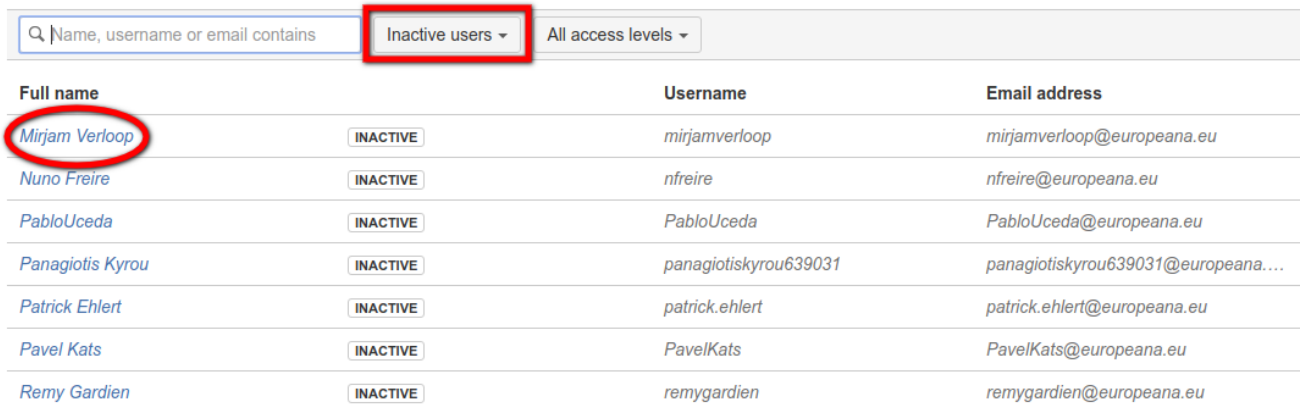
Read in the Assembla user file `data/assembla/:space/users.csv` and create the Jira users if they do not already exist.

```
$ ruby 08-jira_import_users.rb # => data/jira/:space/jira-users.csv
```

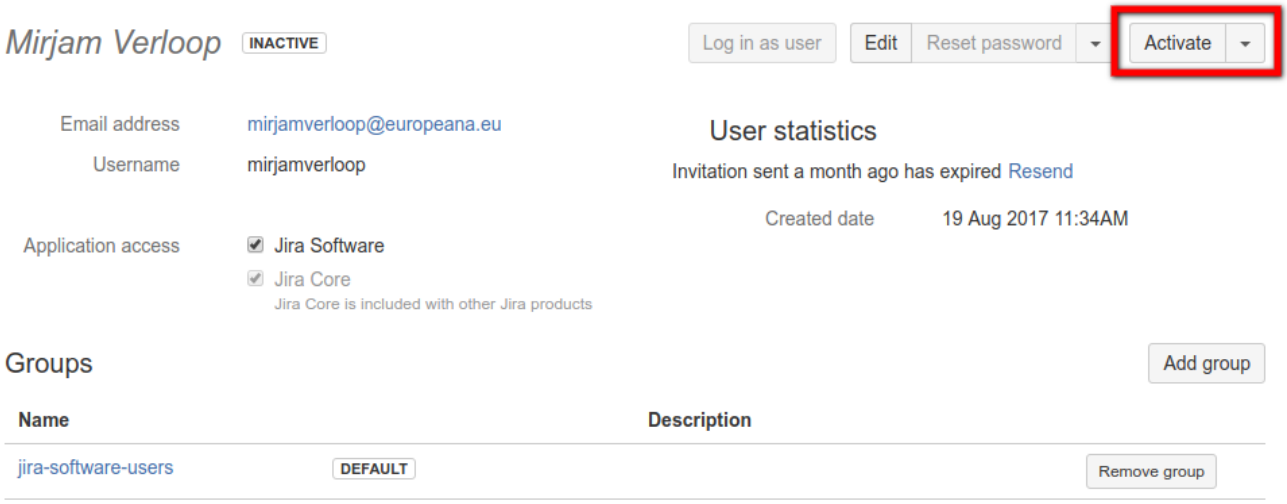
Make sure that all of the users have been activated by going into the admin dashboard user page. In the hosted version this should be the default, however in the cloud version you will need to change each user manually.

If this is the case, you will be given a list of those users that need to be activated.

Go to the Admin User Management:



and after clicking on the username click on the [Activate]-button:



All new user are assigned by default to the `jira-software-users` group only. So do not forget to restore the orignal Assembla admin users permissions by also assigning them to the `jira-administrators` group.

The following user:

- unknown.user@example.org

as defined in the `.env` file as `JIRA_API_UNKNOWN_USER`.

A sanity check will also be made to ensure that the admin user defined as `JIRA_API_ADMIN_USER` in the `.env` file actually exists, is activated and belongs to both the `site-admins` and the `jira-administrators` groups. Otherwise, an error message is generated explaining that this needs to be corrected.

NOTE: Initially the users are created with the `password` equal to their username. This is needed in order for the migration to succeed because of certain user permissions required. Do NOT change until after the migration has been completed.

IMPORTANT: At the end of the import you may be given a warning that certain users need to activate before continuing. Do NOT forget to do this as later actions requiring these users may fail.

Download attachments

Before the attachments can be imported, they must first be downloaded to a local directory after which they can be imported into Jira.

This is accomplished by executing the following command:

```
$ ruby 09-jira_download_attachments.rb # => data/jira/:space/jira-
attachments-download.csv
```

The downloaded attachments are placed in the `data/jira/:space/attachments` directory with the same filename, and the meta information is logged to the file `data/jira/:space/jira-attachments-download.csv` containing the following columns:

```
created_at|assembla_ticket_id|jira_ticket_id|filename|content_type
```

which is used to import the attachments into Jira in the following section. A check is made if the file already exists in order to avoid name collisions.

Note that in Jira images are treated as attachments and can be accessed that way via `[[image:IMAGE|NAME]]`.

Important: this step needs to be done before importing tickets (next section) in order that the markdown for embedded attachment (images) will work correctly.

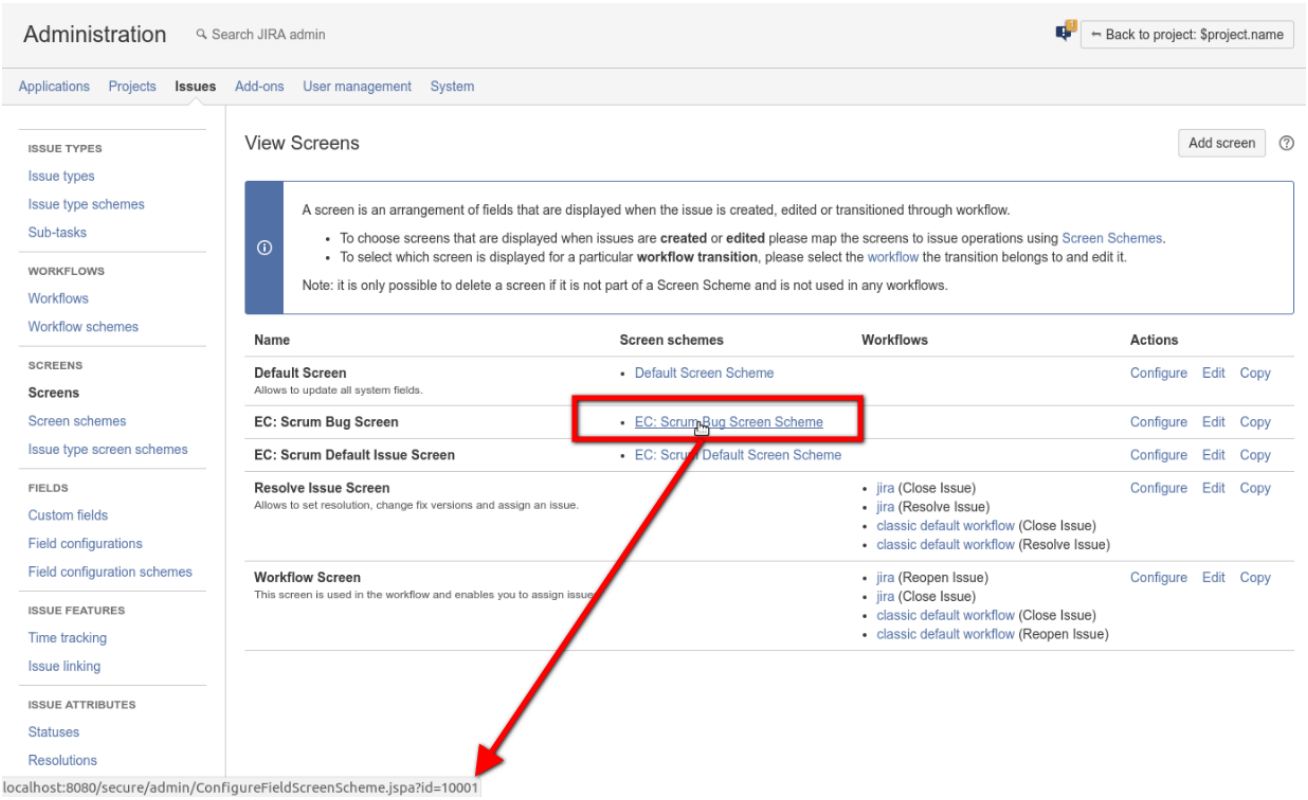
Create custom fields

This step is very important, so do not skip it. You are now ready to create the Jira custom fields, so execute the following command:

```
$ ruby 10-jira_create_custom_fields.rb screen_id1 screen_id2
```

The values of `screen_id1` and `screen_id2` are found by going to the screens page on the admin dashboard at `JIRA_API_HOST/secure/admin/ViewFieldScreens.jspa` and clicking on the project `Bug Screen Scheme` and `Default Screen Scheme` links respectively.

You will be taken to the page whose url `JIRA_API_HOST/secure/admin/ConfigureFieldScreenScheme.jspa?id=10001` indicates the screen id, in this example `id=10001`.

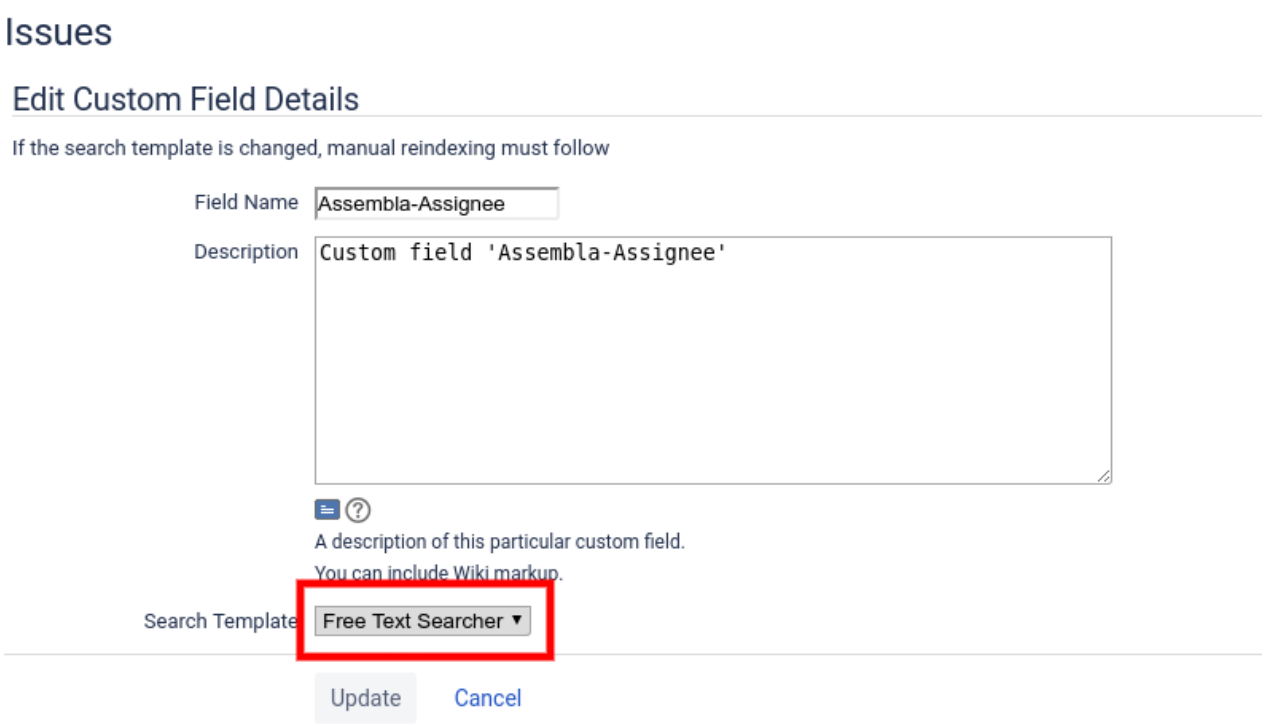


This scripts scans the current Jira custom fields and creates the Assembla fields `Assembla-xxx` which are missing. After that the fields are assigned to the relevant screens given by `screen_id1` and `screen_id2`.

```
POST /rest/api/2/screens/{screenId}/tabs/{tabId}/fields
{
  "fieldId": "summary"
}
def jira_add_field(screen_id, tab_id, field_id)
  ...
end
```

Once all of the custom fields have been created, you want to make sure that a `free text searcher` search template is selected so that the custom field can be sorted in Jira.

Go to the `Issues Custom Fields` page and click on the right of the given row to go to the `Edit Custom Fields Details`.

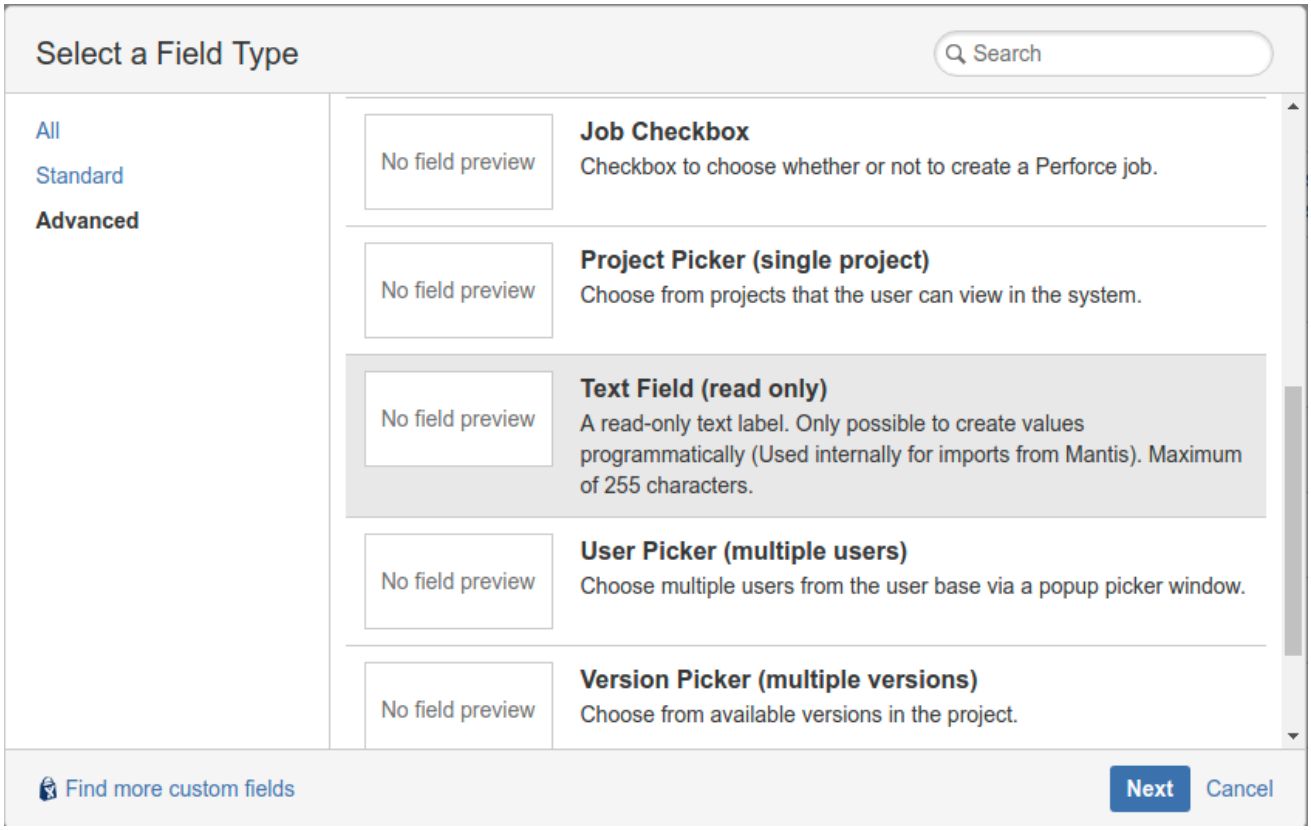


Errors

If for one reason or the other the script fails, you will need to define manually the following custom fields (text field read-only):

- Assembla-Id
- Assembla-Status
- Assembla-Milestone
- Assembla-Reporter
- Assembla-Assignee
- Assembla-Completed
- Assembla-Estimate
- Assembla-Worked
- Assembla-Remaining
- Assembla-(custom-field)

where `Assembla-(custom-field)` is defined by `ASSEMBLA_CUSTOM_FIELD=custom-field` in the `.env` configuration file.

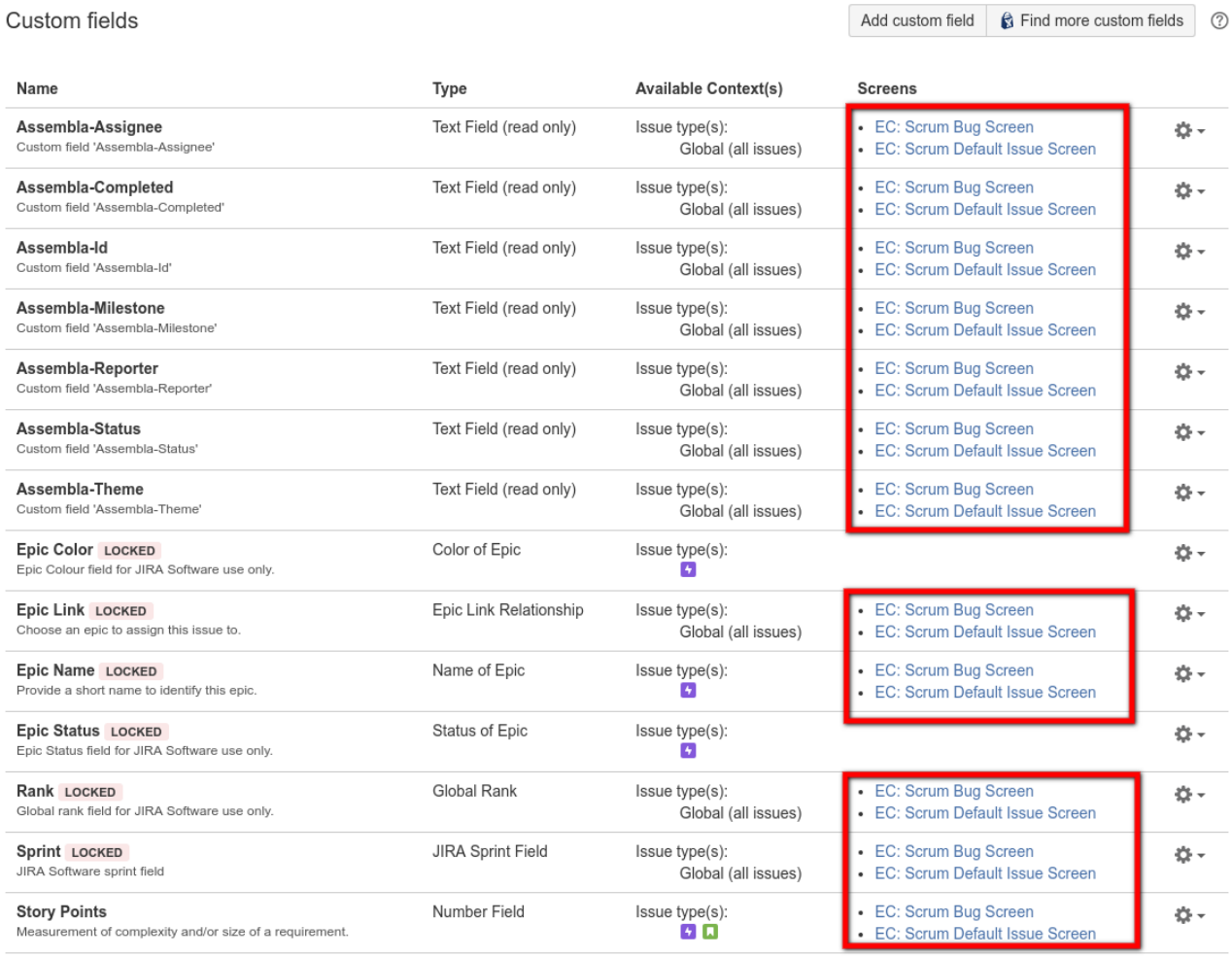


and assign them to the following screens:

- Bug Screen
- Default Issue Screen

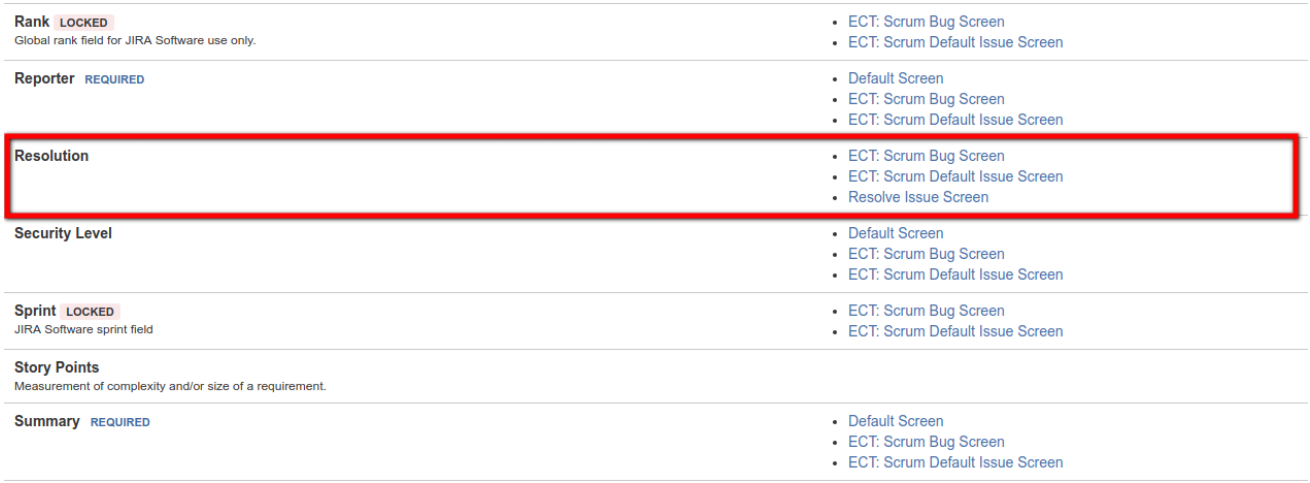
Additionally the following already existing custom fields need to be assigned the the same screens:

- Epic Link
- Epic Name
- Rank
- Sprint
- Story Points



On the `View Field Configuration Page` ensure the same for:

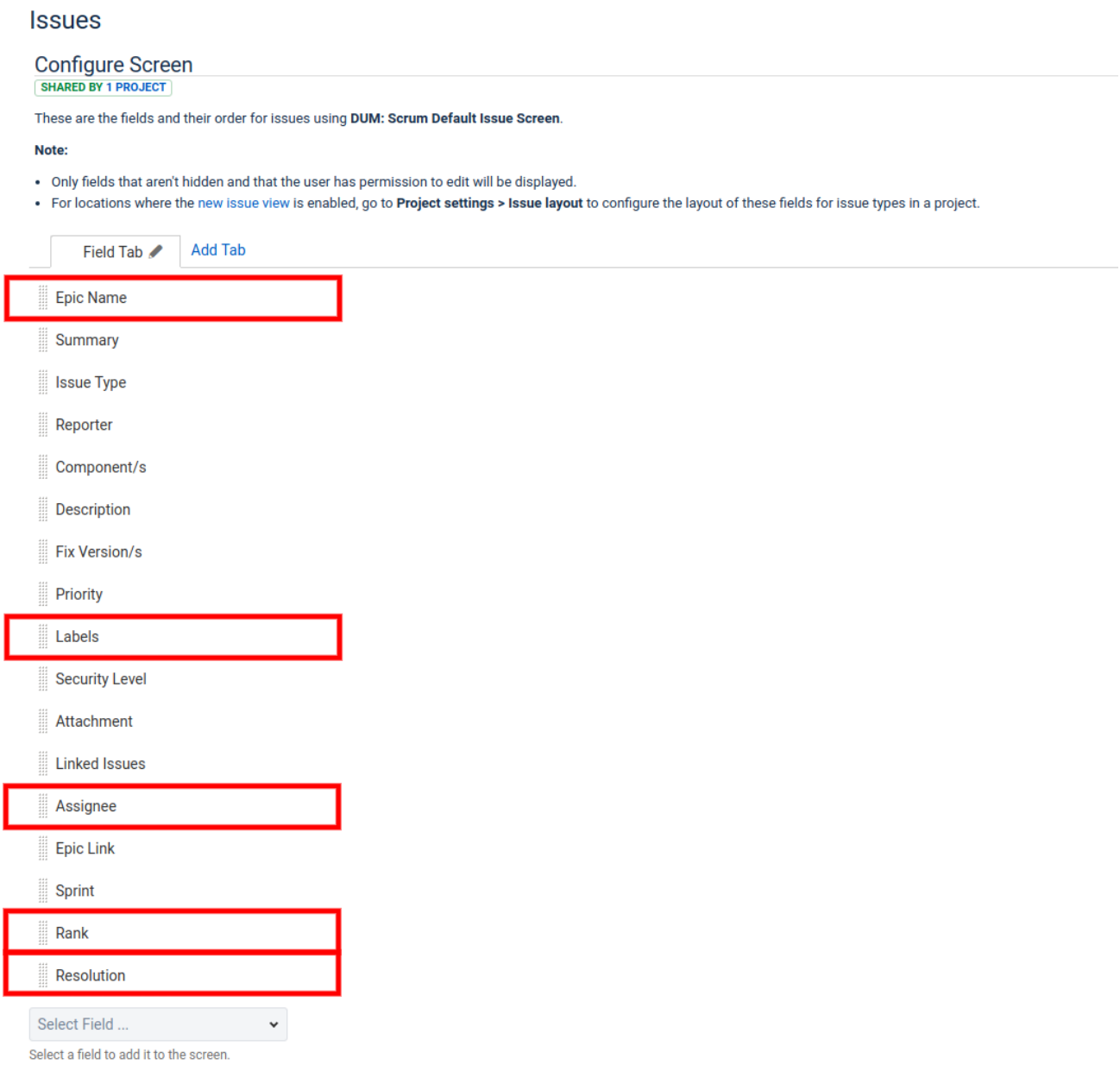
- Resolution



The same applies to the `Configure Screen Page` for BOTH the Scrum Bug and Scrum Default Issue pages, so include the following additional (default) fields:

- Epic Name
- Labels
- Assignee

- Rank
- Resolution



Import custom fields

Assembla allows the use of a number of user-defined field types, namely: `List`, `Team List`, `Numeric` and `Text`.

These need to be mapped properly to the relevant Jira custom fields implemented as Jira plugins `com.atlassian.jira.plugin.system.customfieldtypes:<type>` as follows:

Assembla type	Jira plugin	Searcher key
List	select	multiselectsearcher
Team List	userpicker	userpickergroupsearcher
Numeric	float	exactnumber

Assembla type	Jira plugin	Searcher key
Text	textfield	textsearcher

```
POST /rest/api/2/screens/{screenId}/tabs/{tabId}/fields
{
  "name": name,
  "description": description,
  "type": type,
  "searcherKey": searcherKey
}
```

Execute the following script to have this done:

```
$ ruby 11-jira_import_custom_fields.rb
```

If any custom fields fail to be created, a list will be generated which you can use to fix manually to the Jira project, something like this:

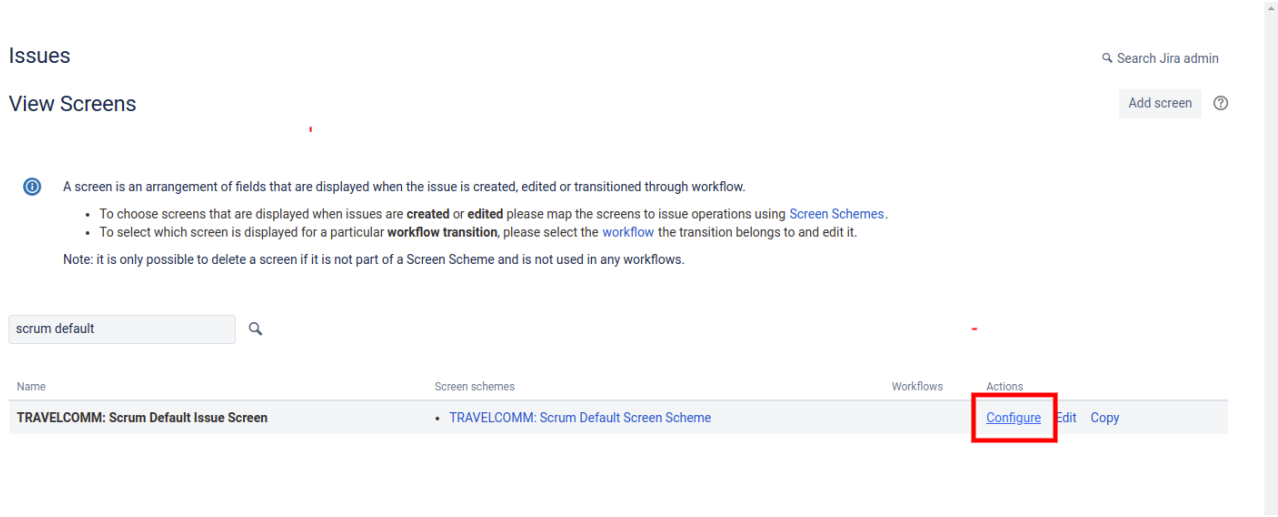
```
IMPORTANT: the following custom JIRA fields MUST be linked to the Scrum
Default and Scrum Bug screens.
* Coverage => type='List'
* Rates => type='List'
* Explanation => type='Text'

IMPORTANT: The following custom JIRA fields are LISTS and you MUST
configure them and add the given options.
* Coverage => ["Low", "Medium", "High"]
* Rates => ["1", "2", "3", "5", "8"]
```

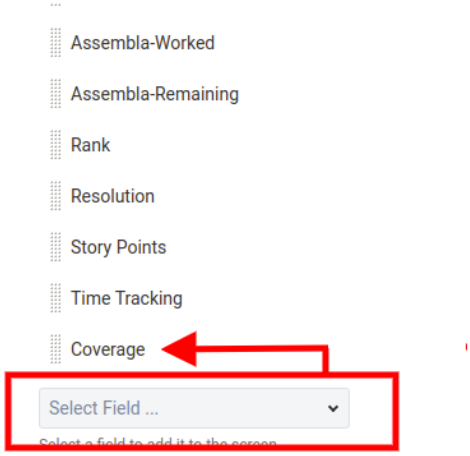
So clearly we have two IMPORTANT actions to take care of manually before we continue to the next step.

1. The first action is to link the listed custom fields to the two screens mentioned above.

Select the given screen name.



In the issues configure screen at the very bottom you will find a select field in which you can enter the name of the given custome field which needs to be added to the sreen..

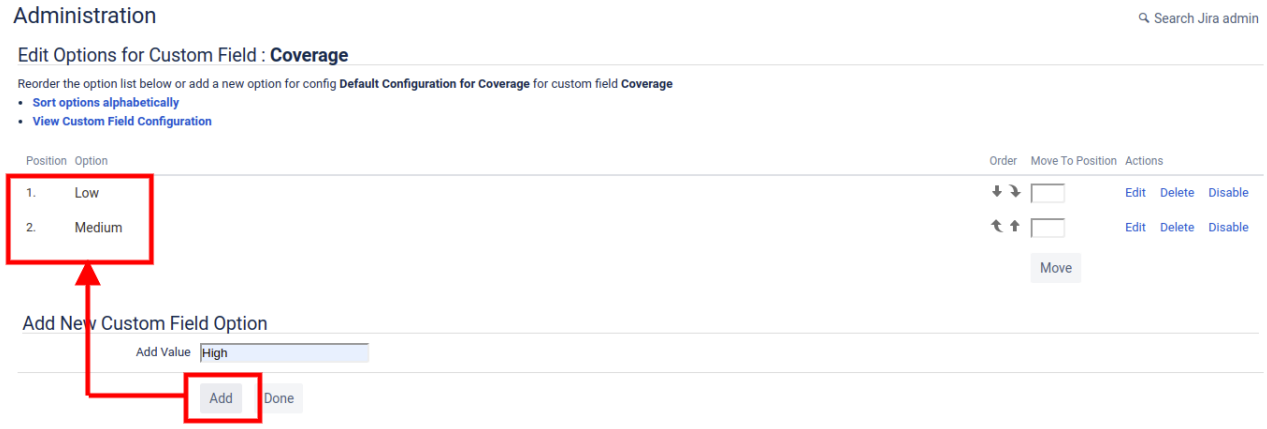


For every listed custome field, please repeat this twice, once for the scrum default issue screen and once for the scrum bug screen.

2. The second action is to configure the the listed custom fields and add the options shown.

Select the given custom field.

Configure it to include the correct item(s).



Import tickets

Alright, this is the moment we've all been waiting for! It's time to import the Assembla tickets and create the matching Jira issues. Here we go.

```
POST /rest/api/2/issue
{
  create: {},
  fields: {
    project: { id: project_id },
    summary: summary,
    issuetype: { id: issue_type[:id] },
    assignee: { name: assignee_name },
    reporter: { name: reporter_name },
    priority: { name: priority_name },
    labels: labels,
    description: description,
    ...
    customfield_assembla_id: ticket_number,
```

```
    customfield_assembla_custom: custom_field,
    customfield_assembla_status: status_name,
    customfield_assembla_milestone: milestone[:name],
    customfield_rank: story_rank, # hosted only

    customfield_assembla_reporter: UNKNOWN_USER, # if reporter is missing
    customfield_assembla_assignee: '',           # if assignee cannot be
assigned issues
    customfield_epic_name: EPIC_NAME,            # if issue type is epic
    parent: { id: parent_id },                   # if issue type is sub-
task
    ...
  }
}
```

Custom fields are also handled accordingly:

```
if custom_field
  assembla_custom_field = "Assembla-#{ASSEMBLA_CUSTOM_FIELD}"
  payload[:fields]["#
{@customfield_name_to_id[assembla_custom_field]}".to_sym] = custom_field
end
```

Now you are ready to import all of the tickets. Execute the following command:

```
$ ruby 12-jira_import_tickets.rb # => data/jira/:space/jira-tickets.csv
```

Results are saved in the output file `data/jira/:space/jira-tickets.csv` with the following columns:

```
jira_ticket_id|jira_ticket_key|project_id|summary|issue_type_id|issue_type_nam
 \
reporter_name|priority_name|status_name|labels|description|assembla_ticket_id|
 \
custom_field|milestone_name|story_rank
```

During the conversion, any differences between the original Assembla ticket description and the newly created Jira issue description is recorded in the `data/jira/:space/jira-tickets-diffs.csv` file. This is a good place to look so you can verify that indeed the markdown conversion produced the expected results.

An additional output file `data/jira/:space/jira-ticket-links.csv` is created which contains those embedded ticket links that could not be resolved. This is used in the following step.

Once completed, check if there are any failed ticket imports where in the `results` column a value of `NOK` is indicated. If present, you can create the Jira issue manually.

Note: it is not possible for the original reporter (creator) of the Assembla ticket to be able to create a new issue, this is only allowed for the admin user, e.g. `headers = JIRA_HEADERS_ADMIN`.

Errors

You might receive an error about a certain issue type that cannot be found. For example `Spike`. This is because you did not create the needed issue types. Please follow instructions in the preparations section above very carefully, and then rerun the `07-jira_get_info.rb` script.

Another error message is `Field 'field-name' cannot be set. It is not on the appropriate screen, or unknown`. This is because either a custom field has not been created or the custom field has not been added to the required screen. See previous section and follow instructions carefully.

Update ticket links

In the ticket summary and description, ticket links `#123` need to be converted to the relevant Jira issue links `PRJ-456`, which can only be done AFTER all the tickets have been imported.

The output file `data/jira/:space/jira-ticket-links.csv` generated in the previous step is used as the input.

Run the following command in order to do this:

```
$ ruby 13-jira_update_ticket_links.rb
```

Note: for one reason or another, not all Assembla links point to valid tickets (deleted, moved or whatever), so these will be marked as invalid by strikethru, e.g. -#123-.

Import comments

```
POST /rest/api/2/issue/{issueIdOrKey}/comment
{
  body: "comments go here..."
}
```

Now you are ready to import all of the comments. Execute the following command:

```
$ ruby 14-jira_import_comments.rb # => data/jira/:space/jira-comments.csv
```

Results are saved in the output file `data/jira/:space/jira-comments.csv` with the following columns:

```
jira_comment_id|jira_ticket_id|assembla_comment_id|assembla_ticket_id|user_log
```

During the conversion, any differences between the original Assembla ticket comments and the newly created Jira issue comments is recorded in the `data/jira/:space/jira-comments-diffs.csv` file. This is a good place to look so you can verify that indeed the markdown conversion produced the expected results.

Since there are so many more comments than tickets, this usually takes the longest by far of all the scripts. My experience using a normal Internet connection to a Jira Cloud instance is that I can import around 60-65 comments per hour.

So assuming we have 2000 tickets with on average 5 comments per ticket, there will be a total of 10,000 comments which will take about 2 hours and 45 minutes.

If `JIRA_API_SKIP_EMPTY_COMMENTS` is true, then only non-empty comments will be imported, e.g. ignore Assembla history stuff. This will make the process go a bit faster, for those impatient folks in the crowd.

Note: we allow the original creators of the Assembla comments to be able to create the new Jira comments, therefore retaining ownership.

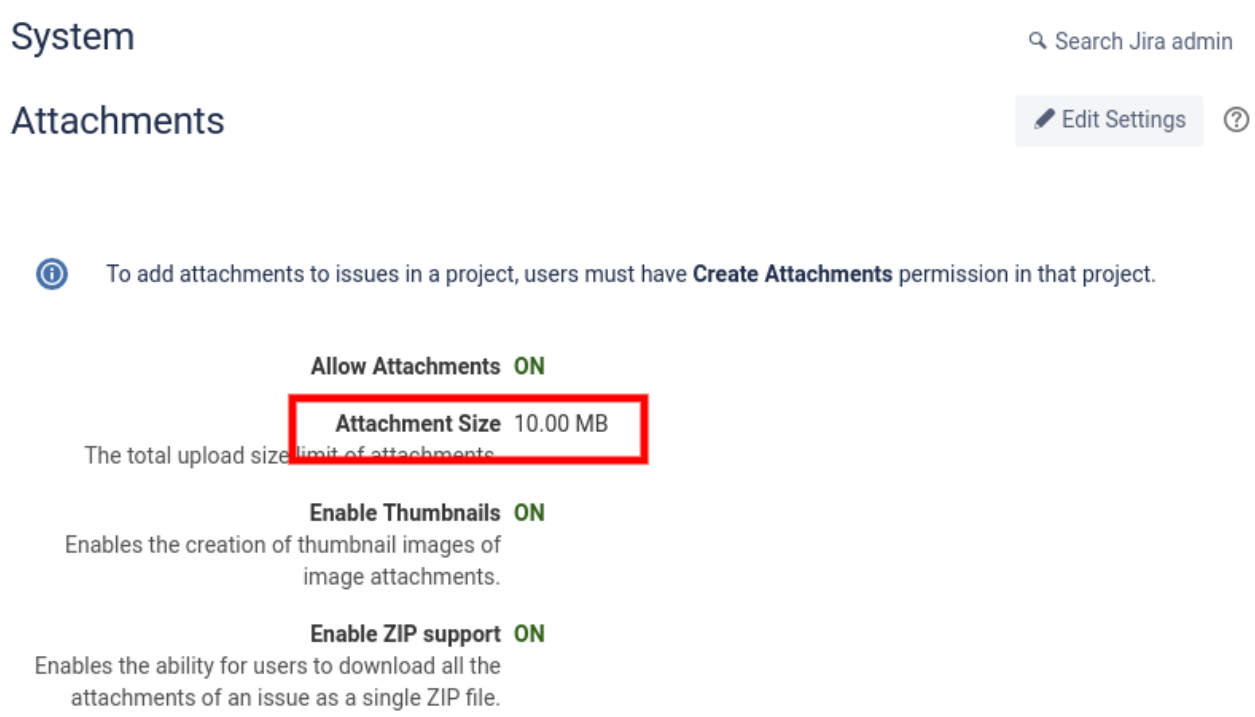
Import attachments

```
curl -D- -u admin:admin -X POST -H "X-Atlassian-Token: no-check" -F
"file=@myfile.txt" \
    api/2/issue/{issueIdOrKey}/attachments
```

Now you are ready to import all of the attachments that were downloaded earlier. Execute the following command:

```
$ ruby 15-jira_import_attachments.rb [restart_offset] # => \
    data/jira/:space/jira-attachments-import-ok.csv
    data/jira/:space/jira-attachments-import-nok.csv
```

Make sure that the admin settings for attachment size is large enough to allow all of the largest attachments to be uploaded.



When completed, don't forget to restore the size to the original value.

Note: The Jira server sometimes has problems processing attachments too quickly and might return an error. In that case, just restart the command and pass it the offset where you want to restart from.

IMORTANT: For the cloud version, the import might fail initially with a `401 Unauthorized` error. Try changing the admin login, logging out and then logging back in again. Hopefully it will now work.

I was able to get things working by defining the following headers:

```
auth = Base64.strict_encode64(admin_email + ':' + admin_password)
headers = { 'Authorization': "Basic #{auth}", 'X-Atlassian-Token': 'no-
check' }
```

Once this script has completed, check out the `jira-attachments-import-nok.csv` file and recover the failed attachments by manually adding them to the indicated Jira issue.

See: Atlassian Community Ticket.

Another note: we allow the original creators of the Assembla attachments to be able to create the new Jira attachments, therefore retaining ownership.

When the migration is completed, one may have a look at the `jira-attachments-import-nok.csv` file and decide whether the failed attachments can be recovered.

Update attachment links

Now that we have imported the attachments, we can convert the Assembla markdown format

```
[[file:attachment_id|filename]]
```

into the Jira markdown format

```
[filename|JIRA_API_BASE/secure/attachment/attachment_id/filename]
```

These markdown links can appear in both the `issue description` or in the `comment body` fields.

You will now need to execute the following script:

```
$ ruby 16-jira_update_attachment_links.rb
```

Update ticket status

Now you are ready to update the Jira tickets in line with the original Assembla state. Execute the following command:

```
$ ruby 17-jira_update_status.rb # => data/jira/:space/jira-update-
status.csv
```

If there are any status types which are missing, the script will abort and display a list of status names that you will have to add manually to Jira.

Important: the Jira API requests MUST be made with an Authorization Header constructed with the `reporter_name` (issue creator), otherwise a `403 Forbidden` error will be returned.

Update ticket associations

For the default Assembla associations the relationship names are:

#	Name	Ticket2	Ticket1
0	Parent	is parent of	is child of
1	Child	is child of	is parent of
2	Related	related to	

#	Name	Ticket2	Ticket1
3	Duplicate	is duplication of	
4	Sibling	is sibling of	
5	Story	is story	is subtask of
6	Subtask	is subtask of	is story
7	Dependent	depends on	
8	Block	blocks	

or in understandable spoken word:

```
0 - Parent (ticket2 is parent of ticket1 and ticket1 is child of ticket2)
1 - Child  (ticket2 is child of ticket1 and ticket2 is parent of ticket1)
2 - Related (ticket2 is related to ticket1)
3 - Duplicate (ticket2 is duplication of ticket1)
4 - Sibling (ticket2 is sibling of ticket1)
5 - Story (ticket2 is story and ticket1 is subtask of the story)
6 - Subtask (ticket2 is subtask of a story and ticket1 is the story)
7 - Dependent (ticket2 depends on ticket1)
8 - Block (ticket2 blocks ticket1)
```

For the default Jira issue link types we have:

Name	Inward	Outward
Blocks	is blocked by	blocks
Cloners	is cloned by	clones
Duplicate	is duplicated by	duplicates
Relates	relates to	relates to

```
POST /rest/api/2/issueLink
{
  type: {
    name: name
  },
  inwardIssue: {
    id: ticket1_id
  },
  outwardIssue: {
    id: ticket2_id
  }
}
```

However, since Jira already takes care of a number of issue links during issue creation (story, subtask, etc), we should disable them in the `.env` configuration file like this:

```
ASSEMBLA_SKIP_ASSOCIATIONS=parent,child,story,subtask
```

If for some reason you do not want to do this, simply comment out the line, or if you prefer to skip other Assembla association just edit the line.

Now you are ready to update the Jira tickets to reflect the original Assembla associations. Execute the following command:

```
$ ruby 18-jira_update_association.rb # => data/jira/:space/jira-update-
associations.csv
```

Important: the Jira API requests MUST be made with an Authorization Header constructed with the `reporter_name` (issue creator), otherwise a `403 Forbidden` error will be returned.

Update ticket watchers

```
POST /rest/api/2/issue/{issueIdOrKey}/watchers
'"username"'
```

Now you are ready to convert the Assembla followers list to the Jira issue watchers list. Execute the following command:

```
$ ruby 19-jira_update_watchers.rb # => data/jira/:space/jira-update-
watchers.csv
```

Important: the Jira API requests MUST be made with an Authorization Header constructed with the `username` (watcher), otherwise a `403 Forbidden` error will be returned.

External ticket/comment links

In the Assembla ticket description and comment body, we might have embedded (external) ticket links that have to be converted to the Jira format.

IMPORTANT: In order to be able to resolve links to external projects, all external projects which are sharing this data need to have been migrated up to but NOT including this step. Once all relevant projects have been migrated to this point, then it is possible to proceed.

These tickets can only be resolved using existing dumps files (`data/jira/:space-name/jira-tickets.csv` and `data/jira/:space-name/jira-comments.csv`) from previous migrations that are indicated in the `.env` file as follows:

```
JIRA_API_SPACE_TO_PROJECT=space1-name:project1-key,space2-name:project2-key
```

Only values of `space-name` present in the `JIRA_API_SPACE_TO_PROJECT` parameter in order to be translated into the Jira equivalent.

For links that point to TICKETS, the captured format looks like:

```
BASE = https?://.*?\.assembla\.com/spaces/(:space-name)

BASE/tickets/(:ticket-number)
BASE/tickets/(:ticket-number)-.*#/activity/ticket:
BASE/tickets/(:ticket-number)/details
BASE/tickets/(:ticket-number)-.*/details
BASE/tickets/(:ticket-number)-.*/details#

REGEX = https?:\/\/.*?\.assembla\.com\/spaces\/(.*?)\/tickets\/(\d+)
(?:\-.*)?(?:\?.*\b)?

$1 = space-name
$2 = ticket-number
```

For links that refer to COMMENTS, we have:

```
BASE = https?://.*?\.assembla\.com/spaces/(:space-name)

BASE/tickets/(:ticket-number)/details?comment=(:comment-id)
BASE/tickets/(:ticket-number)-.*/details?comment=(:comment-id)

REGEX = https?:\/\/.*?\.assembla\.com\/spaces\/(.*?)\/tickets\/(\d+).*?\?
comment=(\d+)(?:#comment:\d+)?

$1 = space-name
$2 = ticket-number
$3 => comment-id
```

and then the links are converted like this:

```
issue => /browse/[JIRA_ISSUE_KEY]
comment => /browse/[JIRA_ISSUE_KEY]?focusedCommentId=
[JIRA_COMMENT_ID]&page= \
  com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-
[JIRA_COMMENT_ID]
```

Execute the following command to update all external links:

```
$ ruby 20-jira_update_ext_links.rb => jira-links-external-all.csv
                                      jira-links-external-updated.csv
```

Two output files are generated for reference:

```
jira-links-external-all.csv     => all detected external links are listed
jira-links-external-updated.csv => only those external links actually
updated
```

Check the output file `jira-links-external-all.csv` for the external links that resulted in errors, e.g. result `NOK`. The `message` field will give you the error that the server returned, so that you can hopefully fix this manually.

Move stories to epics

```
POST /rest/agile/1.0/epic/{epicIdOrKey}/issue
{
  "issues": issues
}
```

The Jira stories originally belonging to an epic in Assembla now need to be added to the newly created Jira epic.

In order to do this you need to execute the following command.

```
$ ruby 21-jira_update_epics.rb
```

The results are saved in the `jira-update-epics.csv` output file.

Check this output file for the epics that resulted in errors, e.g. result `NOK`. The `message` field will give you the error that the server returned, so that you can hopefully fix this manually. For example, `Issue 'EC-71' is an epic and therefore cannot be associated to another epic` is a common message.

If errors occur, e.g. `Issue 'EC-71' is an epic and therefore cannot be associated to another epic`, you should run the following recovery script which will attempt to fix most of the problems:

```
$ ruby 22-jira_update_epics_nok.rb
```

The results are saved in the `jira-update-epics_nok.csv` output file, a result of `NOK` meaning that you may attempt to fix it manually with the help of the `message` column giving the error text.

Rank tickets

Only needed for the Jira server type is `cloud`. Since this was not possible during the ticket creation, now is the time to rank the imported issues using the original Assembla values.

```
$ ruby 23-jira_rank_tickets.rb
```

Scrum Board

You are now ready to setup the scrum board, create sprints, and assign issues to the correct sprints as well as the backlog. In the `.env` file, take notice of the following values:

```
JIRA_API_PROJECT_NAME=name
JIRA_API_PROJECT_TYPE=scrum
JIRA_BOARD_NAME=name:Scrum Board Name
```

These will be used as placeholder values below.

Create sprints

When the scrum board was created with the project, all issues are assigned to the project are automatically put in the backlog.

Now you are ready to setup the sprints by executing the following command:

```
$ ruby 24-jira_create_sprints.rb # => data/jira/:space/jira-create-
sprints.csv
```

The issues are redistributed to the sprints they belong to and the most recent sprint is set as the `active` sprint.

If the milestone title is not less than 30 characters, then it will be truncated with an ellipsis before assigning the sprint name to it.

In order to be able to create a sprint, both a start and an end date must be provided.

If no start date is given, then by default a date 2 weeks previous to the end date will be used, and if there is no end date provided, then 2 weeks before the current date will be used.

If no end date is given, then by default a date 2 weeks after the start date will be used, and if there is no start date provided, then 2 weeks after the current date will be used.

Errors

You might receive an `403 Unauthorized` error. If this is the case, go to the Jira application, login as admin and try again.

Update board

The final step after the board and sprints have been created is to copy the Assembla cardwall columns (ticket statuses) to the Jira board and to order the issues by rank as they were in Assembla.

In order to achieve this, execute the following command:

```
$ ruby 25-jira_update_board.rb


GET /rest/agile/1.0/board/{boardId}/configuration
```

At the time of this writing, the Jira API does not yet support creating new columns. Therefore, when the command above is executed you will see some output:

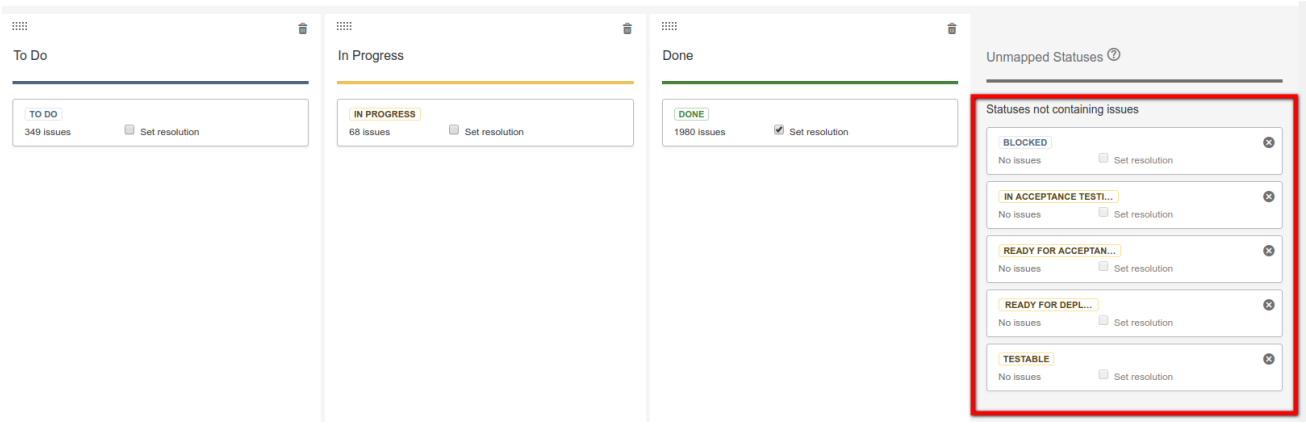
```
Board columns needed: 7
* New => To Do
* In progress => In Progress
* Testable => Testable
* Ready for acceptance => Ready for Acceptance
* In acceptance testing => In Acceptance Testing
* Ready for deploy => Ready for Deploy

Board columns actual: 3
* To Do
* In Progress
* Done
```

Followed by instructions on which columns need to be added manaully with a link showing where this can be done:

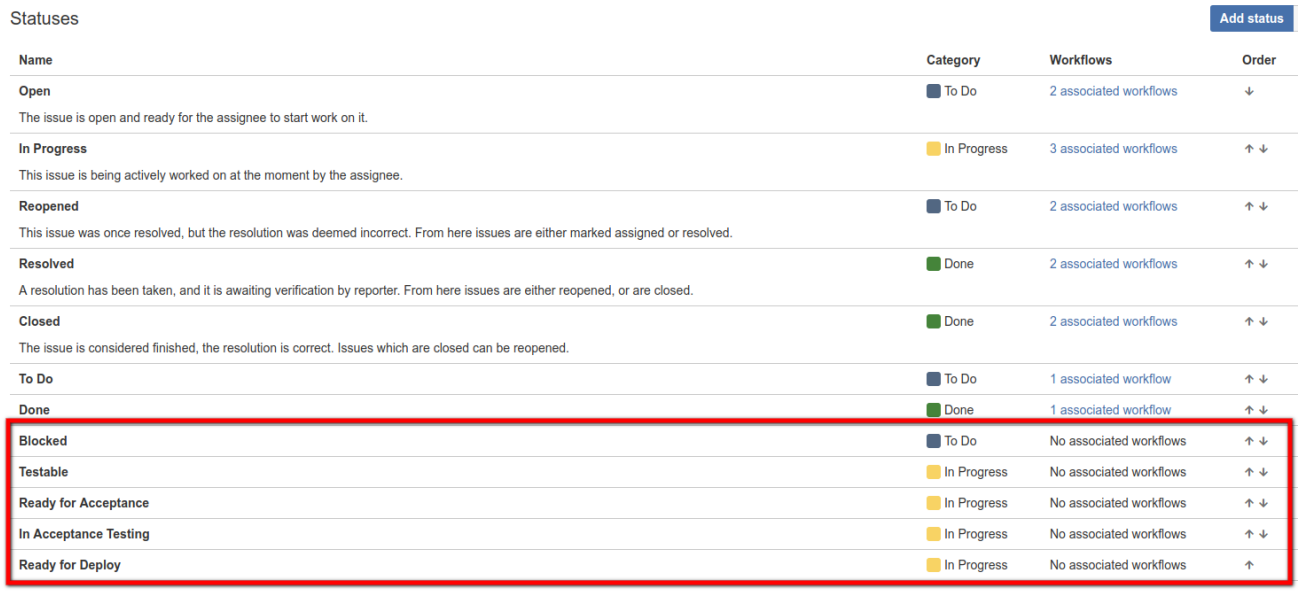
```
Go to Configure 'BOARD_NAME | Column Management' and add the following
columns:
* Testable
* Ready for Acceptance
* In Acceptance Testing
* Ready for Deploy

link: JIRA_API_BASE/secure/RapidView.jspa?rapidView=3&tab=columns
```

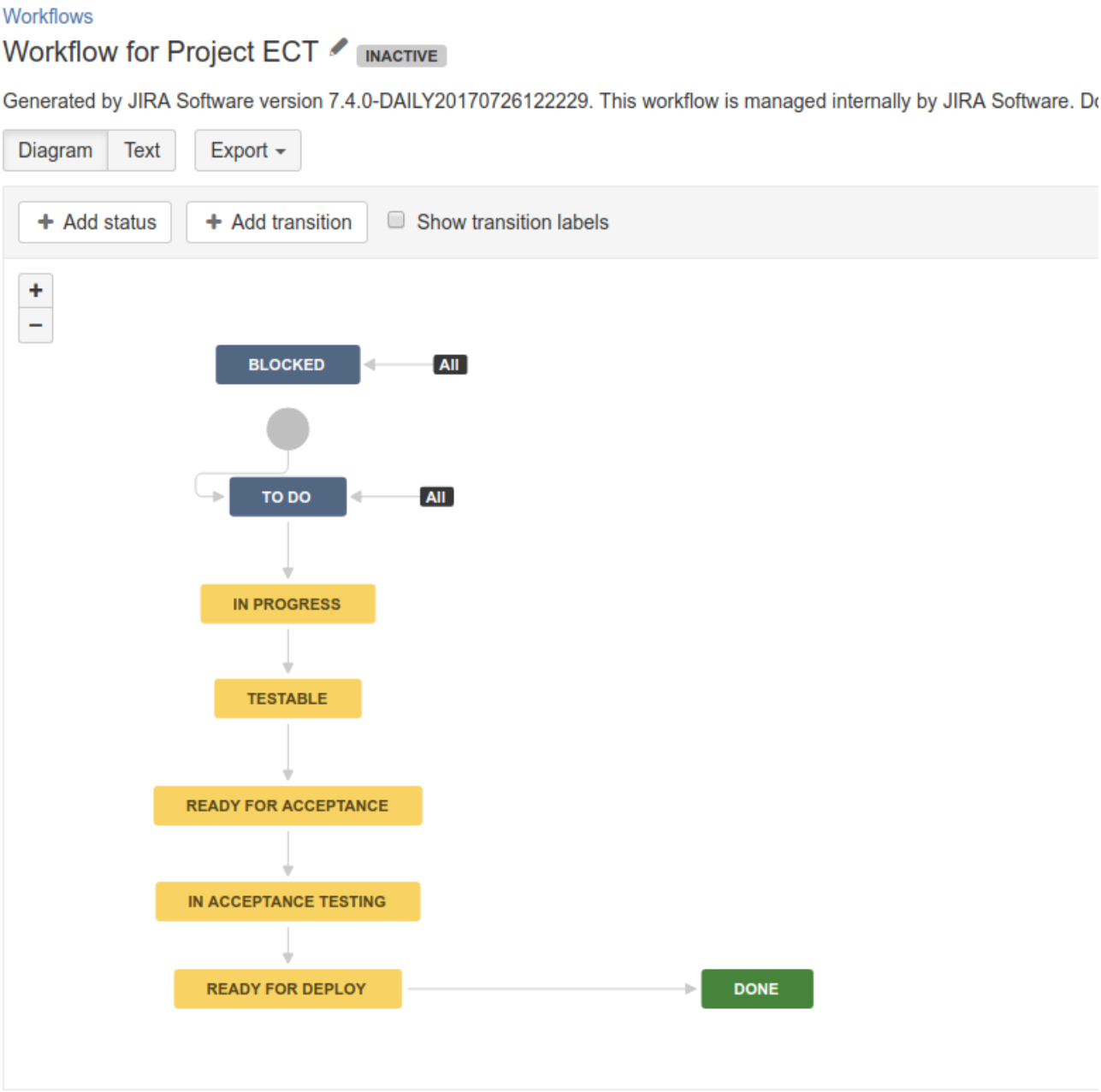


Create statuses

```
JIRA_API_STATUSES=New:To Do,In Progress,Blocked,Testable,Ready for
Acceptance, \
    In Acceptance Testing,Ready for Deploy,Done,Invalid:Done
```



Create workflow



Import Assembla Wiki to Confluence

This is a new script which allows one to take the previously generated Assembla wiki exported `wiki-pages.csv` file, and execute a best-effort import script to a given Confluence space.

The `.env` file has been extended with the following extra items:

```
# --- Confluence settings --- #
CONFLUENCE_API=https://[:company-name].atlassian.net/wiki/rest/api
CONFLUENCE_SPACE=[:space-name]
CONFLUENCE_EMAIL=john.doe@example.org
CONFLUENCE_PASSWORD=secret
```

These values need to be updated to the once you will be using for the Wiki migration.

First create the new Confluence space by going to the `https://[:company-name].atlassian.net/wiki/` page and clicking the `Create Space` -button at the top right of the page.

Select the `Blank space` option and enter the `Space name` corresponding to the value of `[:space-name]` in the `.env` file and optionally the `Space key` if you desire one other than the generated default.

Once the page has been created you can go ahead and run the script:

```
$ ruby 26-wiki_to_confluence.rb
```

The script attempts to do the following tasks:

- upload all pages
- update all page links
- upload all images
- update all image links
- update all markdown page links
- update all markdown url links
- upload all documents
- update all document links
- update all ticket links

Dowloaded documents can be found in the `confluence/documents` directory and the downloaded images in the `confluence/images` directory.

The generated csv-files are saved in the `confluence` directory. These are used within the script but can also be reviewed to detect any warning or errors.

Additionally, a number of (commented out) scripts at the end are provided to assist you with trouble-shooting and analyzing possible missing or incorrect attachment, ticket or image links.

```
check_for_regexes([/#\d+/, /\[.*?\]\(.*?\)/, /<code>.*?<\/code>/])
check_for_header_lines
check_for_tickets
```

The import of wiki pages into confluence involves a complex set of text transformation in which a best effort is attempted to convert the wiki html, markdown and plain text formats into the appropriate Confluence XHTML format.

This is very challenged and there will be some anomolies in the resulting transformations, so one is advised to double-check the results as best as possible.

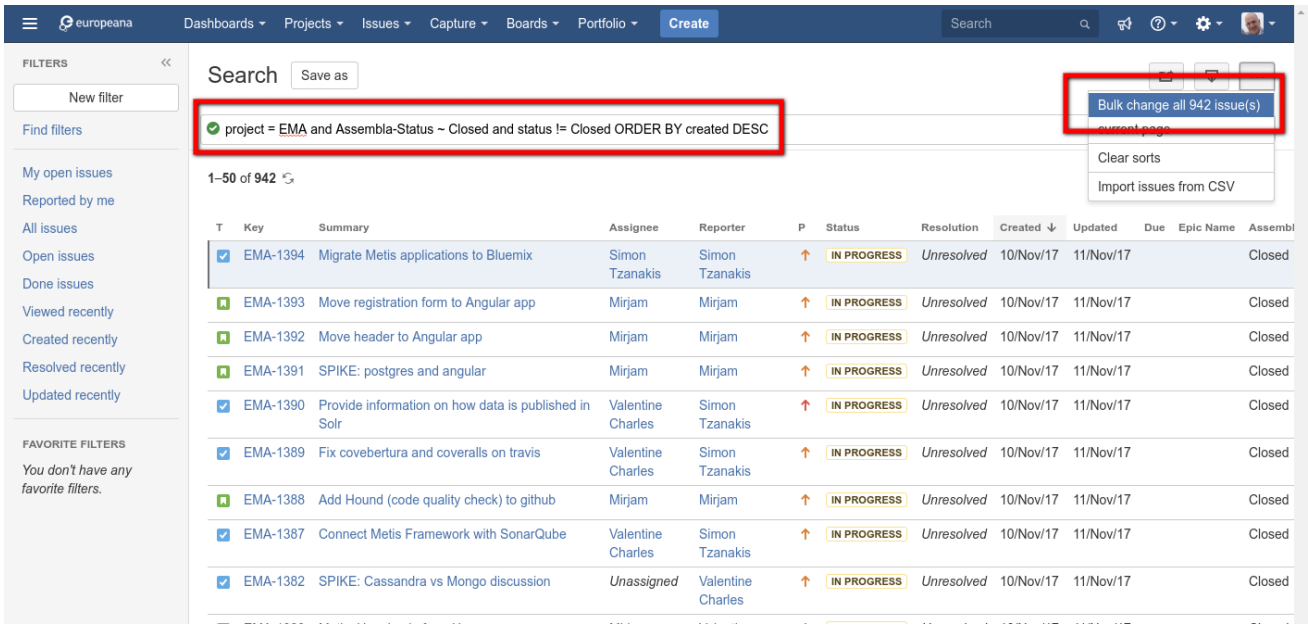
Cleanup

Finally, cleanup actions need to be taken to finish things off.

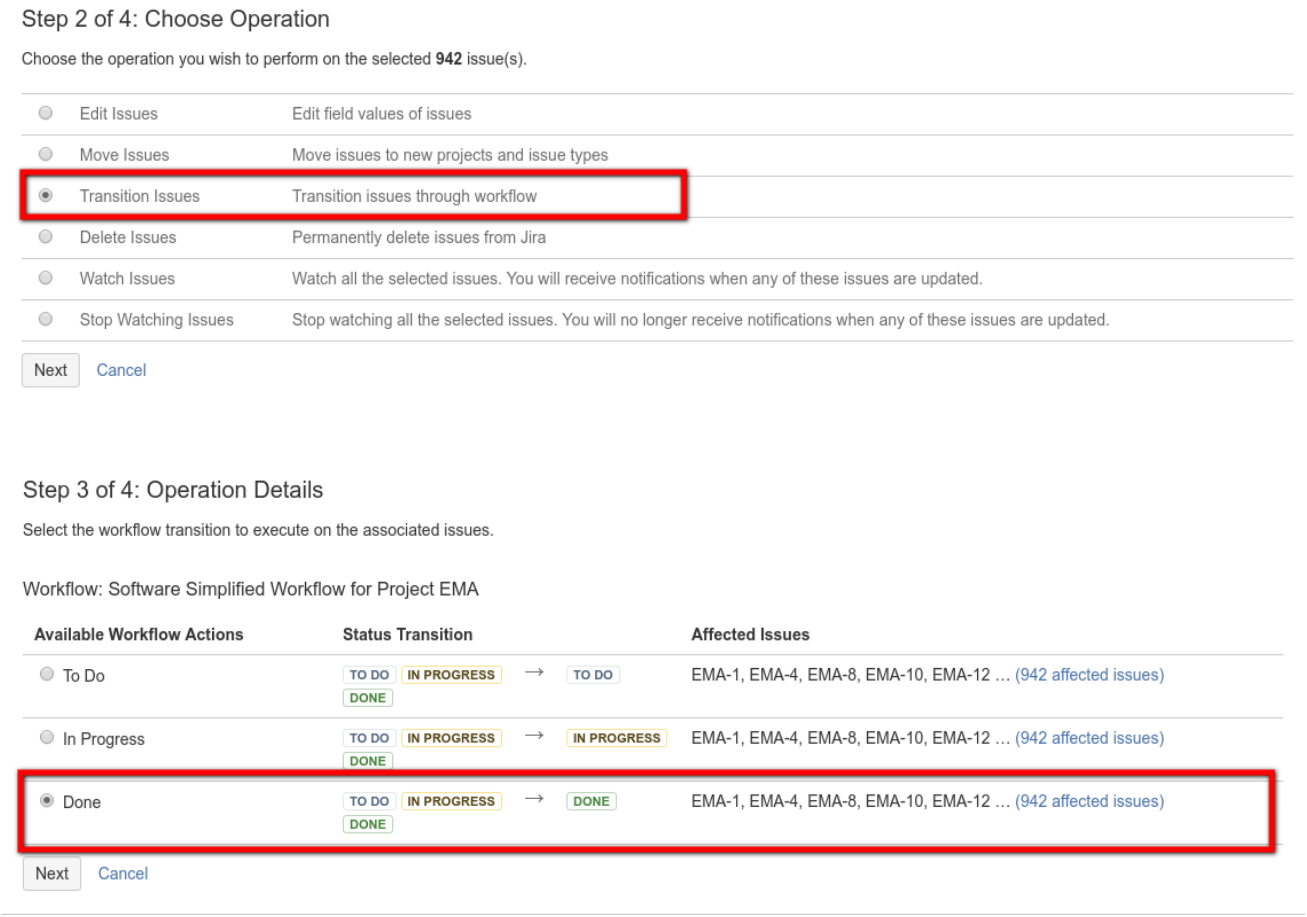
- Deactivate users not needed.
- Give admin rights to relevant users.
- Assign project leads (and give permissions).
- Ask users to change password, check email and create avatar.
- Recover failed attachment uploads listed in `jira-attachments-import-nok.csv`.
- Resolve failed external links listed as `NOK` in `jira-external-links.csv`.
- Recover failed epic updates listed in `jira-update-epics-nok.csv`.
- Use label filters to move issue to correct types, e.g. `bug` might be a label.
- Check that tickets which are spikes are NOT epics Issue 14.
- Make backup of `data` directory including `.env` file for future reference.

You should also double check that the all of the `Assembla-Status` were converted properly to the correct Jira status. If that is not the case, then you can make changes in bulk. For instance, filter on issues where Assembla-Status is `closed` and Jira status is NOT `closed` and by selecting all you can convert them to `Done` in one go.

```
project = PROJECT_KEY and Assembla-Status ~ Done and status != Closed ORDER BY created DESC
```



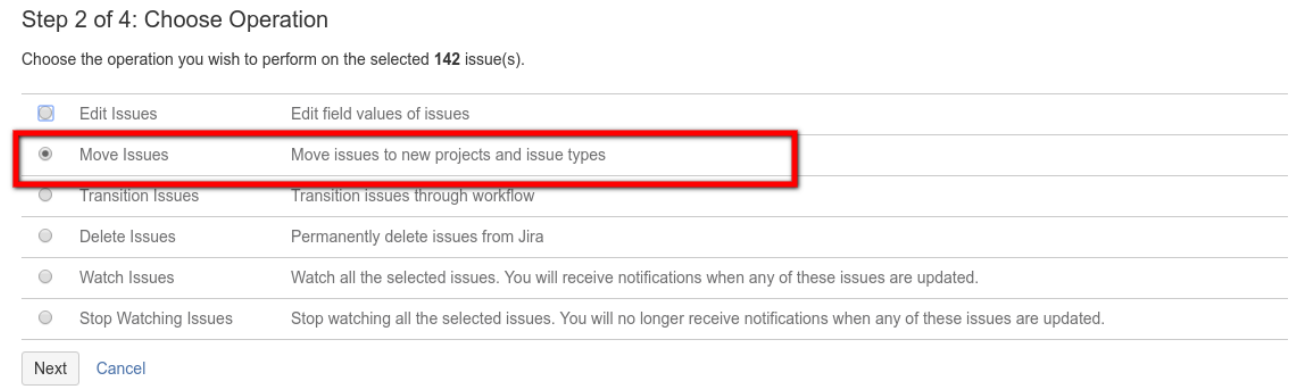
Followed by transition issues to done:

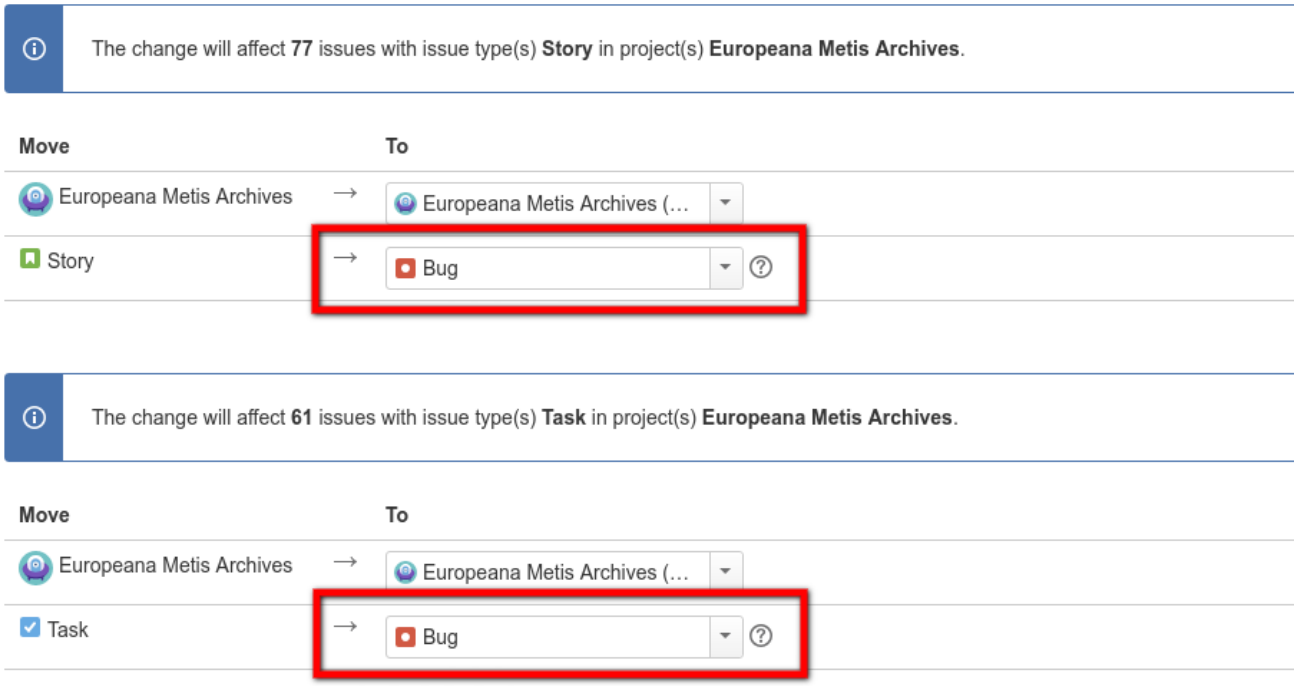


Another example might be selecting all issues with `Assembla-Type` equal to `Bug` to be converted to the Jira `Bug` issue type.

```
project = PROJECT_KEY and Assembla-Type ~ Bug and issuetype != Bug ORDER BY created DESC
```

Followed by move issues:





Checklist

It can be slightly tedious running scripts that take a long time to complete and keeping track of where you stand in the scripts pipeline.

In order to make this easier for you to track, here is a simple checklist where you can sign off each step and remember where you are.

Step	Actions	Item	Dir	Start	Done
01	Assembla	Space	dn		
02	Assembla	Tickets	dn		
03	Assembla	Users	na		
04	Assembla	Reports	na		
05	Jira	Projects	up		
06	Jira	Issue links	na		
07	Jira	General info	na		
08	Jira	Users	up		
09	Jira	Attachments	dn		
10	Jira	Create custom fields	dn		
11	Jira	Import Custom fields	up		

Step	Actions	Item	Dir	Start	Done
12	Jira	Tickets	up		
13	Jira	Ticket links	up		
14	Jira	Comments	up		
15	Jira	Attachments	up		
16	Jira	Attachment links	up		
17	Jira	Status	up		
18	Jira	Associations	up		
19	Jira	Watchers	up		
20	Jira (1)	Ext links	up		
21	Jira	Epics	up		
22	Jira (2)	Epics NOK	up		
23	Jira (3)	Ranking	up		
24	Board	Sprints	up		
25	Board	Update	up		
26	Wiki	Import	up		
27	Cleanup	See list	na		

(1) first complete all projects up to this point before continuing (in order to ensure that all of the external links are resolved correctly).

(2) only if errors occurred in the previous step.

(3) only for cloud server.

Output files

As mentioned previously, during each step of the migration pipeline, the script will generate output in the form of CSV files to capture the data at that given moment.

Assembla

In the `data/assembla/:space` directory:

- documents.csv

- milestones-all.csv
- report-tickets.csv
- report-users.csv
- spaces.csv
- space-tools.csv
- tags.csv
- ticket-associations.csv
- ticket-attachments.csv
- ticket-comments.csv
- tickets.csv
- tickets-custom-fields.csv
- tickets-statuses.csv
- ticket-tags.csv
- user-roles.csv
- users.csv
- wiki-pages.csv

Jira

In the `data/jira/:space` directory:

- jira-attachments-download.csv
- jira-attachments-import-nok.csv
- jira-attachments-import-ok.csv
- jira-comments.csv
- jira-comments-diffs.csv
- jira-issuelink-types.csv
- jira-issue-types.csv
- jira-links-external-all.csv
- jira-links-external-updated.csv
- jira-priorities.csv
- jira-projects.csv
- jira-resolutions.csv
- jira-roles.csv
- jira-serverinfo.csv
- jira-sprints.csv
- jira-statuses.csv
- jira-ticket-links.csv

- jira-tickets-associations.csv
- jira-tickets.csv
- jira-tickets-diffs.csv
- jira-tickets-status-updates.csv
- jira-tickets-watchers.csv
- jira-update-epics.csv
- jira-update-epics-nok.csv
- jira-users.csv

Ticket field conversions

Most of the ticket fields are converted from Assembla to Jira via a one-to-one mapping. These fields are indicated as **bold** below:

Assembla ticket fields:

- id
- **number**
- **summary**
- **description**
- **priority** (1 - Highest, 2 - High, 3 - Medium, 4 - Low, 5 - Lowest)
- **completed_date**
- component_id (deprecated)
- **created_on**
- permission_type
- **importance** (Sorting criteria for Assembla Planner) => 10104 Rank
- is_story (true or false, if true hierarchy_type = 2)
- **milestone_id** => 10103 Sprint
- **tags**
- followers
- **notification_list**
- **space_id**
- **state**
  - 0 - closed, 1 - open
- **status** (new, in progress, blocked, testable, ready for acceptance, in acceptance testing, ready for deploy, done, invalid)
- **story_importance** (1 - small, 4 - medium, 7 - large) => 10105 Story Points (for stories only)
- updated_at

- working_hours
- estimate
- total_estimate
- total_invested_hours
- total_working_hours
- **assigned_to_id**
- **reporter_id**
- **custom_fields**
- **hierarchy_type** (0 - No plan level, 1 - Subtask, 2 - Story, 3 - Epic)
- is_support
- due_date
- picture_url

Jira issue fields:

Default

- **issuetype**
- timespent
- **project**
- fixVersions
- aggregatetimespent
- resolution (done, won't do, duplicate)
- resolutiondate
- workratio
- lastViewed
- **watches**
- thumbnail
- **created**
- **priority** (1 - Highest, 2 - High, 3 - Medium, 4 - Low, 5 - Lowest)
- **labels**
- timeestimate
- aggregatetimeoriginalestimate
- versions
- **issuelinks**
- **assignee**
- **updated**
- **status** (todo, done)

- components
- **issuekey**
- timeoriginalestimate
- **description**
- timetracking
- security
- **attachment**
- aggregatetimeestimate
- **summary**
- **creator**
- **subtasks**
- **reporter**
- aggregateprogress
- environment
- duedate
- progress
- **comments**
- votes
- worklog

Custom

- 10000 Development
- 10001 Team
- 10002 Organizations
- **10003** Epic Name
- 10004 Epic Status
- 10005 Epic Color
- 10006 Epic Link
- **10007** Parent Link
- 10100 [CHART] Date of First Response
- 10101 [CHART] Time in Status
- 10102 Approvals
- **10103 Sprint**
- **10104 Rank**
- **10105 Story Points**
- 10108 Test sessions
- 10109 Raised during

- 10200 Testing status
- 10300 Capture for Jira user agent
- 10301 Capture for Jira browser
- 10302 Capture for Jira operating system
- 10303 Capture for Jira URL
- 10304 Capture for Jira screen resolution
- 10305 Capture for Jira jQuery version
- **10400 Assembla**

Confluence

In the `data/confluence` directory:

- check-tickets.csv
- created-pages.csv
- created-pages-nok.csv
- links.csv
- uploaded-documents.csv
- uploaded-images.csv
- wiki-documents.csv
- wiki-pages-fixed.csv
- wiki-tickets.csv

Downloaded files can be found in the following directories:

- data/confluence/documents
- data/confluence/images

Authorization

Depending on the server type, the authorization is handled slightly differently. For the hosted server the user_login and password (same as user_login) are used, whereas for the cloud we use the user_email and password.

```
def headers_user_login(user_login, user_email)
  cloud = JIRA_SERVER_TYPE == 'cloud'
  user_login_or_email = cloud ? user_email : user_login
  user_password = user_login
  base64_encoded = Base64.strict_encode64(user_login_or_email + ':' +
user_password)
  {
    'Authorization': "Basic #{base64_encoded}",
    'Content-Type': 'application/json'
```

```
    }
  end
```

where `user_login` is either the `JIRA_API_ADMIN_USER` for global configurations (create/update projects, issue types, issue link types and sprints) or the `reporter_name` (issue creator) for updating certain issue specific attributes (status, associations, watchers, issue description and comment body).

Associations

The Assembly associations are converted into Jira issue links.

```
0 - Parent (ticket2 is parent of ticket1 and ticket1 is child of ticket2)
1 - Child  (ticket2 is child of ticket1 and ticket2 is parent of ticket1)
2 - Related (ticket2 is related to ticket1)
3 - Duplicate (ticket2 is duplication of ticket1)
4 - Sibling (ticket2 is sibling of ticket1)
5 - Story (ticket2 is story and ticket1 is subtask of the story)
6 - Subtask (ticket2 is subtask of a story and ticket1 is the story)
7 - Dependent (ticket2 depends on ticket1)
8 - Block (ticket2 blocks ticket1)
```

See: http://api-docs.assembla.cc/content/ref/ticket_associations_fields.html

Statuses and states

The Assembla ticket statuses are: `new`, `in progress`, `blocked`, `testable`, `ready for acceptance`, `in acceptance testing`, `ready for deploy`, `done` and `invalid`.

An Assembla ticket can have two states: `0 - closed` (done or invalid) and `1 - open` (all others).

The Jira statuses are: `todo` and `done`. On creation, all Jira tickets are set initially to `todo` by default.

The possible transitions for this initial `todo` state are `start progress => in progress` and `done => done`.

During the migration, Assembla tickets that are marked as `closed` will result in Jira issues marked as `done` with resolution set to `fixed` for Assembla ticket status `done` and `won't fix` for Assembla ticket status `invalid`.

For Assembla tickets marked as `in progress` the imported Jira issue will be set to `in progress`.

IMPORTANT: all the other statuses will be ignored unless the administrator modifies the workflow for the given Jira project to include them explicitly.

The names of these newly defined transitions MUST be the same as the Assembla status names in order for the status migration to work properly.

Story points

The `story_importance` field for Assembla tickets is ONLY used for `story` type Jira issues.

Components

For the time being components have not yet been implemented.

According to the Assembla API Documentation: `Ticket components API is deprecated. Please use custom fields.`

Markdown

The Assembla markdown syntax is different from Jira Markdown. Therefore, the certain markdown notations will need to be translated as follows.

Equivalent (no changes required)

```
h1. TITLE
h2. TITLE
*bold*
_italic_
Bullet list
Numbered list
Numbered - Bullet list
```

Ignore (will be ignored and passed through unchanged)

```
Wiki links
[[ticket:NUMBER]]
```

Reformat (will be reformatted into Jira markdown)

```
#TICKET_NR => JIRA_TICKET_KEY
[[image:IMAGE]] => !name(IMAGE)|thumbnail!
[[image:IMAGE|text]] => !name(IMAGE)|thumbnail!
@NAME => [~accountid:id]
[[user:NAME]] => [~accountid:id]
[[user:NAME|text]] => [~accountid:id]
@INLINE_CODE@ => {{INLINE_CODE}} (monospaced)
<code>INLINE_CODE</code> => {{INLINE_CODE}} (monospaced)
```

```
[[url:URL|TEXT]] => [TEXT|URL]
[[url:URL]] => [URL|URL]
<pre><code> code-snippet </code></pre> => {code:java} code-snippet {code}
[[file:attachment_id|filename]] =>
[filename|JIRA_API_BASE/secure/attachment/attachment_id/filename]
```

Code blocks

In Assembla a block of code looks like this:

```
<pre><code>
code-snippet
</code></pre>
```

which will be transformed into Jira format like this:

```
{code:java}
code-snippet
{code}
```

Note that the images will have original or thumbnail sizes depending on the value of `JIRA_API_IMAGES_THUMBNAIL` in the `.env` file.

So for example:

```
JIRA_API_IMAGES_THUMBNAIL=description:false,comments:true
```

would insert original size images in the Jira issue description and thumbnail images in the Jira issue comments (which happens to be the default).

For the content available in the ticket summaries, descriptions and comments we have:

```
[summary, description, comments].each do |content|
  content = reformat_markdown(content, opts)
end
```

where `reformat_markdown` will do the following global substitutions:

```
gsub(/<pre><code>/i,'{code:java}').
gsub(/<\/code><\/pre>/i,'{code}').
gsub(/\[\[url:(.*?)\|(.*?)\]\]/i, '[\2|\1]').
gsub(/\[\[url:(.*?)\]\]/i, '[\1|\1]').
gsub(/<code>(.*?)<\/code>/i,'{{\1}}').
gsub(/@([^@]*)@( |$)/, '{{\1}}\2').
gsub(/@([a-z.-_]*)/i) { |name| markdown_name(name, user_ids) }.
```

```
gsub(/\[\[user:(.*?)(\|(.*?))?\]\]/i) { |name| markdown_name(name,
user_ids) }.
gsub(/\[\[image:(.*?)(\|(.*?))?\]\]/i) { |image| markdown_image(image,
images, content_type) }
```

Trouble-shooting

- Strange permission errors when creating tickets, adding watchers, etc. This is more than likely because the user defined by `JIRA_API_ADMIN_USER` does not belong to the `jira-administrators` group.
- Ticket import error `key='issuetype', reason='The issue type selected is invalid.'` . Go to the project issue types scheme, edit and ensure that issue type is included in the list, e.g. spike.
- Error `403 Unauthorized` . Go to the Jira application, login as admin and try again.
- Import users to the cloud fails for some user for some mysterious reason (500 Internal Server Error). This happens sometimes, just restart the script. It should recover and continue where it last failed. If the problem keeps repeating itself, just keep on retrying the script until you make your way through the complete list.
- A `403 Forbidden` or `401 Unauthorized` error is returned. Ensure that the Authorization header is correct. if that doesn't work, log into your Atlassian account id.atlassian.com and try changing your password. There are some known problems with a recent cloud upgrade, see Atlassian Community Ticket, and certain extra actions must be taken. If problem persists, make sure that you are physically logged in to the hosted or cloud instance.
- Error `User cannot be assigned issues.` Activate, login as user and then deactivate.
- If issue is an epic then the epic name custom field is required.
- XSRF check failed => This is a known bug.
- Ticket or other import fails with the error message `Field 'field-name' cannot be set. It is not on the appropriate screen, or unknown` . Ensure that the custom field 'field-name' has been created and assigned to the required screens (see above). If this doesn't work, make sure that the user named in the authorization header has enough rights to make these changes.
- Error `key='customfield_10100 (Assembla-Completed)', reason='Operation value must be a number'` , ensure that the custom field is the correct type: text field read-only.

To do

With such a complicated tool, there will always be some loose ends and/or additional work to be done at a later time. Hopefully in the not so distant future, I'll have some time to tackle one or more of the following items:

- Must have: Update readme screenshots and relevant screen associations, e.g. only `Scrum Default Issue Screen` is required. Issue 6
- Bug: Ticket type 'Spike' is converted to an Epic. Issue 14
- Nice to have: Ignore sprints with no issues Issue 27
- Nice to have: Split large imports into smaller batches Issue 26
- Nice to have: Support multiple Assembla custom fields instead of just one. Issue 2
- Nice to have: Rank tickets (cloud) in batches of fifty instead of individually. Issue 15
- Nice to have: Create Jira board columns in line with the original Assembla cardwall columns (statuses = blocked, testable, ready for acceptance, in acceptance testing, ready for deploy) and populate with the relevant issues. Issue 4
- Nice to have: Allow data dumps to restart with all newer items since last dump, rather than having to start all over again. This is already the case for attachments, but should be possible with tickets, comments, etc. Issue 5
- Nice to have: Assembla tickets with tag `bug` should be converted into Jira issue of type `bug`. Issue 7
- Wish: Allow themes to be converted into Epics (additional .env file option). Currently epics are only created for tickets with summaries that start with 'EPIC:' which in hindsight is probably not the best way of doing this. Issue 3
- Wish: Use a user-defined Jira project template instead of requiring the user to define stuff manually. Issue 9
- Wish: Assign original authors as creators of tickets (this might not be possible) Issue 10
- Refactor: Merge the recovery script `18-jira_update_epics_nok.rb` into `18-jira_update_epics.rb` Issue 16
- Refactor: cleanup code, remove duplication, fix rubocop warnings, and make more object-oriented using classes. Issue 11

References

- Assembla
  - Homepage
  - API Reference
  - Markdown
- Jira
  - Homepage
  - JIRA Server platform REST API reference
  - JIRA Cloud REST API Reference
  - Markdown
  - Upgrade to Atlassian Account

Support

Do you require assistance with the migration or need some new functionality that is not yet part of this package?

No worries, I can certainly help you out.

Feel free to contact me!

Author

Kiffin Gish < kiffin.gish@planet.nl >

Gishtech => Advanced Software Development for the Web

"You're never too old to learn new stuff..."

Releases 7

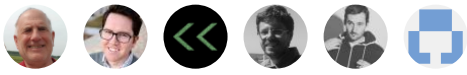
v2.0 Latest
37 minutes ago

+ 6 releases

Packages

No packages published
Publish your first package

Contributors 6



Languages

Ruby 79.6% **JavaScript** 19.7% **Shell** 0.7%