IssueSyncTool

v. 0.0.1

Tran Duy Ngoan

01.11.2024

Contents

1 Introduction **1**
1.1 Use Cases . . . 1
1.2 Benefits . . . 1

2 Description **3**
2.1 Tool features . . . 3
2.2 Tool usage . . . 3
2.3 JSON Configuration File . . . 4
2.3.1 Source and Destination Platforms . . . 4
2.3.2 Tracker Configurations . . . 4
2.3.3 User Mapping . . . 4

3 rtc_client.py **6**
3.1 Function: get_xml_tree . . . 6
3.2 Class: RTCClient . . . 6
3.2.1 Method: get_complexity_link . . . 6
3.2.2 Method: get_filedAgainst . . . 7
3.2.3 Method: get_info_from_url . . . 7
3.2.4 Method: login . . . 7
3.2.5 Method: get_workitem . . . 8
3.2.6 Method: update_workitem . . . 8
3.2.7 Method: update_workitem_state . . . 8
3.2.8 Method: update_workitem_action . . . 9
3.2.9 Method: create_workitem . . . 9

4 sync_issue.py **10**
4.1 Function: write_csv_files . . . 10
4.2 Function: process_cli_argument . . . 10
4.3 Function: process_configuration . . . 10
4.4 Function: process_new_issue . . . 11
4.5 Function: process_sync_issues . . . 11
4.6 Function: SyncIssue . . . 12
4.7 Class: Logger . . . 12
4.7.1 Method: config . . . 12
4.7.2 Method: log . . . 12
4.7.3 Method: log_warning . . . 13
4.7.4 Method: log_error . . . 13

5 tracker.py **14**
5.1 Class: Status . . . 14
5.1.1 Method: normalize_issue_status . . . 14
5.1.2 Method: get_native_status . . . 14
5.2 Class: Ticket . . . 15
5.2.1 Method: update . . . 15
5.2.2 Method: is_synced_issue . . . 15
5.3 Class: TrackerService . . . 15
5.3.1 Method: connect . . . 15
5.3.2 Method: get_ticket . . . 16
5.3.3 Method: get_tickets . . . 16
5.3.4 Method: create_ticket . . . 16
5.3.5 Method: exclude_issue_by_condition . . . 16
5.3.6 Method: get_story_point_from_labels . . . 17
5.3.7 Method: time_estimate_to_story_point . . . 17
5.4 Class: JiraTracker . . . 17
5.4.1 Method: connect . . . 18
5.4.2 Method: get_ticket . . . 18
5.4.3 Method: get_tickets . . . 18
5.4.4 Method: create_ticket . . . 18
5.4.5 Method: update_ticket . . . 19
5.4.6 Method: get_story_point . . . 19
5.4.7 Method: create_label . . . 19
5.5 Class: GithubTracker . . . 20
5.5.1 Method: connect . . . 20
5.5.2 Method: get_tickets . . . 20
5.5.3 Method: get_ticket . . . 20
5.5.4 Method: create_ticket . . . 21
5.5.5 Method: update_ticket . . . 21
5.5.6 Method: create_label . . . 21
5.6 Class: GitlabTracker . . . 22
5.6.1 Method: connect . . . 22
5.6.2 Method: get_ticket . . . 22
5.6.3 Method: get_tickets . . . 23
5.6.4 Method: create_ticket . . . 23
5.6.5 Method: update_ticket . . . 23
5.6.6 Method: get_story_point . . . 24
5.6.7 Method: create_label . . . 24
5.7 Class: RTCTracker . . . 24
5.7.1 Method: connect . . . 24
5.7.2 Method: get_ticket . . . 25
5.7.3 Method: get_tickets . . . 25
5.7.4 Method: get_plannedFor . . . 25
5.7.5 Method: update_ticket_state . . . 26
5.7.6 Method: create_ticket . . . 26
5.7.7 Method: update_ticket . . . 26

5.8 Class: Tracker 26
5.8.1 Method: create 27
5.8.2 Method: get_support_trackers 27

6 user.py **28**
6.1 Class: User 28
6.2 Class: UserManagement 28
6.2.1 Method: get_user 28

7 Appendix **29**

8 History **30**

Chapter 1

Introduction

The **IssueSyncTool** is a command-line utility designed to streamline issue synchronization across various tracking systems, including **GitHub**, **Jira**, **GitLab** and **IBM RTC**.

Its primary objective is to automate and simplify the integration and synchronization of issues between these platforms, enabling efficient tracking and planning for teams that work with multiple tools.

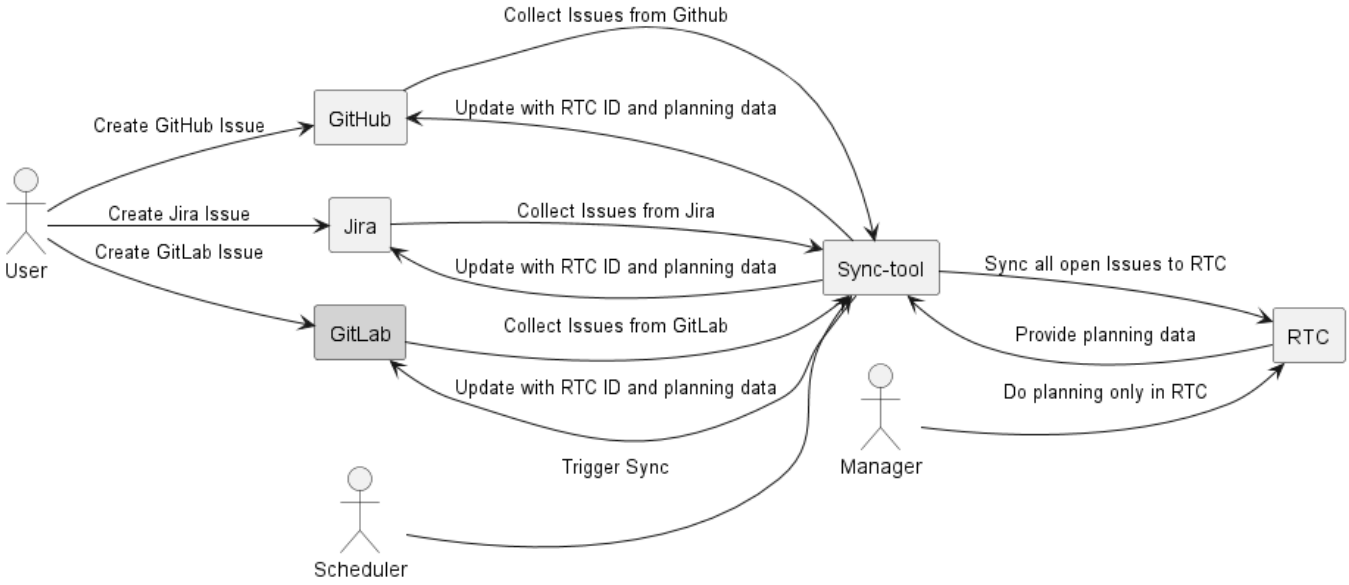

Figure 1.1: Tool's use case

1.1 Use Cases

- **Multi-Tool Teams:** For teams using a combination of GitHub, Jira, and RTC for issue management, this tool acts as a bridge to consolidate data.
- **Planning and Reporting:** Synchronization ensures managers and stakeholders have a centralized view of issues for effective planning.
- **Automated Workflows:** With scheduled triggers, the tool eliminates manual synchronization efforts, saving time and reducing errors.

1.2 Benefits

- **Automation:** Reduces manual synchronization overhead.
- **Consistency:** Ensures data integrity across platforms.
- **Customizable:** Flexible configurations to suit various project needs.

- **Centralized Planning:** Aligns all issues with RTC, the central planning tool.

Chapter 2

Description

2.1 Tool features

The **IssueSyncTool** facilitates seamless integration and synchronization between multiple issue tracking platforms. The main operations include:

1. Configuration Parsing
   - Reads the JSON configuration file to understand the synchronization scope and behavior.
2. Issue Collection
   - Fetches issues from GitHub, Gitlab and Jira based on the specified conditions.
   - Uses user mappings to ensure issues are associated correctly across platforms.
3. Issue Update
   - Updates the source issues with RTC IDs and planning data after synchronization.
4. Synchronization to RTC
   - Creates or updates work items in RTC with the collected issues.
   - Includes planning data provided by RTC.

2.2 Tool usage

Use below command to get tools's usage:

```
IssueSyncTool -h
```

The tool's usage should be showed as below:

```
usage: IssueSyncTool (Tickets Sync Tool) [-h] --config CONFIG [--dryrun] [--csv] [-v]

IssueSyncTool sync ticket|issue|workitem between tracking systems such as Github Issue, ↩
    ↪ JIRA and IBM RTC

optional arguments:
  -h, --help       show this help message and exit
  --config CONFIG  path to configuration json file
  --dryrun         if set, then just dump the tickets without syncing
  --csv            if set, then store the sync status to csv file sync_status.csv
  -v, --version    version of the IssueSyncTool
```

Sample command to run **IssueSyncTool** with the configuration JSON file and save sync status as csv file:

```
IssueSyncTool --config your-config-file --csv
```

2.3 JSON Configuration File

The tool uses a JSON configuration file to define synchronization behavior. Below is an explanation of the sample configuration:

2.3.1 Source and Destination Platforms

```
{
   "source": ["github", "gitlab", jira"],
   "destination": ["rtc"]
   ...
}
```

This configuration specifies GitHub and Jira as sources and RTC as the destination for synchronization.

2.3.2 Tracker Configurations

```
{
   ...
   "tracker": {
      "github": {
         "project" : "test-fullautomation",
         "token": "your-PAT",
         "repository": [
            "python-jsonpreprocessor"
         ],
         "condition": {
            "state": "open",
            "exclude": {
               "assignee": "empty",
               "labels": "0.13.1"
            }
         }
      },
      ...
   }
   ...
}
```

Above code is sample configuration of Github tracker which contain the information about:

- **Project:** Github project `"test-fullautomation"`
- **Token:** A personal access token for authentication.
- **Repositories:** List of repositories to collect issues from.
- **Condition:** Define the condition (query) to collect the issues.
  - `"state"` : Syncs only issues in the specified state, e.g., `"open"` .
  - `"exclude"` : Specifies negative conditions. For example:
    - `"assignee": "empty"` : Excludes issues with no assignee.
    - `"labels": "0.13.1"` : Excludes issues labeled `"0.13.1"` .

The other tracker can be configured as the same way.

2.3.3 User Mapping

User mapping ensures that the correct user is assigned in the synchronization process across different platforms. In the configuration file, each user is mapped to their corresponding accounts across GitHub, Jira, Gitlab and RTC. This mapping helps to ensure that the right assignee is applied to issues in the appropriate tracker system.

- The `user` section of the configuration file specifies the mapping between the users' names in GitHub, Gitlab, Jira and RTC.
- This ensures that the correct user is set as the assignee in each platform when syncing issues.
- For instance, when syncing an issue from Jira to RTC, the tool will automatically assign the same user (as per the mapping) to the issue in RTC.
- If the user has different usernames across platforms (e.g., "githubUser" in GitHub, "jiraUser" in Jira, and "rtcUser" in RTC), the tool ensures the correct mapping is applied so that all systems reflect the same assignee.

Example configuration:

```
{
   ...
   "user": [
      {
         "name": "Tran Duy Ngoan",
         "github": "ngoan1608",
         "jira": "ntd1hc",
         "gitlab": "ntd1hc",
         "rtc": "ntd1hc"
      },
      ...
   ]
}
```

In this example:

- The user `Tran Duy Ngoan` is mapped to `ngoan1608` in GitHub, `ntd1hc` in Jira, and `ntd1hc` in RTC.
- When syncing issues between GitHub, Jira, and RTC, the tool ensures that issues assigned to `ngoan1608` in GitHub and `ntd1hc` in Jira will be assigned to `ntd1hc` in RTC, ensuring consistent user data across all platforms.

Chapter 3

rtc_client.py

3.1 Function: get_xml_tree

Parse xml object from file.
Arguments:

- `file_name`
  / *Condition*: required / *Type*: str /
  The name of the file to parse.
- `bdtd_validation`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  Whether to validate the XML against a DTD.

Returns:

- `oTree`
  / *Type*: etree.ElementTree /
  The parsed XML tree.

3.2 Class: RTCClient

Imported by:

```
from IssueSyncTool.rtc_client import RTCClient
```

Client for interacting with RTC (Rational Team Concert).

3.2.1 Method: get_complexity_link

Get the complexity link for the specified story point.
Arguments:

- `story_point`
  / *Condition*: required / *Type*: int /
  The story point value.
- `project_id`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project ID.

Returns:

- `complexity_link`
  / *Type*: str /
  The complexity link for the specified story point.

3.2.2 Method: get_filedAgainst

Get the filed against URL for the specified file against name.

Arguments:

- `fileAgainst_name`
  / *Condition*: required / *Type*: str /
  The file against name.
- `project_id`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project ID.

Returns:

- `fileAgainst_url`
  / *Type*: str /
  The filed against URL.

3.2.3 Method: get_info_from_url

Get the specified information from the URL.

Arguments:

- `url`
  / *Condition*: required / *Type*: str /
  The URL to request.
- `info`
  / *Condition*: required / *Type*: str /
  The information to retrieve.

Returns:

- `info_value`
  / *Type*: str /
  The retrieved information value.

3.2.4 Method: login

Authenticate and establish a session with RTC.

3.2.5 Method: get_workitem

Get a work item by its ID.

Arguments:

- `ticket_id`
  / *Condition*: required / *Type*: str /
  The ID of the work item.

Returns:

- `work_item`
  / *Type*: dict /
  The work item data.

3.2.6 Method: update_workitem

Update a work item with the specified attributes.

Arguments:

- `ticket_id`
  / *Condition*: required / *Type*: str /
  The ID of the work item.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  The attributes to update.

Returns:

- `None`

3.2.7 Method: update_workitem_state

Update the state of a work item.

Arguments:

- `ticket_id`
  / *Condition*: required / *Type*: str /
  The ID of the work item.
- `current_state`
  / *Condition*: required / *Type*: str /
  The current state of the work item.
- `new_state`
  / *Condition*: required / *Type*: str /
  The new state of the work item.

Returns:

- `None`

3.2.8 Method: update_workitem_action

Update the state of a work item by performing the specified action.

Arguments:

- `ticket_id`
  / *Condition*: required / *Type*: str /
  The ID of the work item.
- `action`
  / *Condition*: required / *Type*: str /
  The action to perform.

Returns:

- `None`

3.2.9 Method: create_workitem

Create a new work item.

Arguments:

- `title`
  / *Condition*: required / *Type*: str /
  The title of the work item.
- `description`
  / *Condition*: required / *Type*: str /
  The description of the work item.
- `story_point`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  The story point value.
- `file_against`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The file against which to create the work item.
- `assignee`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The assignee of the work item.
- `project_id`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project ID.
- `kwargs`
  / *Condition*: optional / *Type*: dict / *Default*: None /
  Additional keyword arguments for creating the work item.

Returns:

- `workitem_id`
  / *Type*: str /
  The ID of the created work item.

Chapter 4

sync_issue.py

4.1 Function: write_csv_files

Write a list of lines to a CSV file.

Arguments:

- `filename`
  / *Condition*: required / *Type*: str /
  The name of the CSV file.
- `list_line`
  / *Condition*: required / *Type*: list /
  A list of lines to write to the CSV file.

Returns:

(*no returns*)

4.2 Function: process_cli_argument

Process command-line arguments.

Returns:

- `args`
  / *Type*: Namespace /
  The parsed command-line arguments.

4.3 Function: process_configuration

Process the configuration JSON file.

Arguments:

- `path_file`
  / *Condition*: required / *Type*: str /
  The path to the configuration JSON file.

Returns:

- `config`
  / *Type*: dict /
  The configuration dictionary.

4.4 Function: process_new_issue

Process to create new issue on destination tracker and update original issue's title with destination issue's id.

- New issue's description is consist of original issue url and its description.
- Assignee is get from

Arguments:

- `issue`
  / *Condition*: required / *Type*: Issue /
  The original issue object.
- `des_tracker`
  / *Condition*: required / *Type*: TrackerService /
  The destination tracker service.
- `assignee`
  / *Condition*: required / *Type*: User /
  The assignee user object.

Returns:

- `res_id`
  / *Type*: str /
  The ID of the created issue on the destination tracker.

4.5 Function: process_sync_issues

Update source (original) issue due to information from appropriate destination one.

Defined sync attributes:

- 'Title': add issue ID as prefix e.g '[123] Ticket title'when creating on destination tracker
- 'Story point': when planning existing issue on destination tracker
- 'Version': when planning existing issue on destination tracker

Update destination issue due to information from source.

Defined sync attributes:

- 'Status': status is synced from original ticket, not allow to update directly on destination tracker

Arguments:

- `org_issue`
  / *Condition*: required / *Type*: Issue /
  The original issue object.
- `org_tracker`
  / *Condition*: required / *Type*: TrackerService /
  The original tracker service.
- `dest_issue`
  / *Condition*: required / *Type*: Issue /
  The destination issue object.

- `des_tracker`
  / *Condition*: required / *Type*: TrackerService /
  The destination tracker service.

Returns:
(*no returns*)

4.6 Function: SyncIssue

Main function to sync issues between tracking systems.
Returns:
(*no returns*)

4.7 Class: Logger

Imported by:

```
from IssueSyncTool.sync_issue import Logger
```

Logger class for logging messages.

4.7.1 Method: config

Configure Logger class.
Arguments:

- `output_console`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  Write message to console output.
- `output_logfile`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Path to log file output.
- `dryrun`
  / *Condition*: optional / *Type*: bool / *Default*: True /
  If set, a prefix as 'dryrun' is added for all messages.

Returns:
(*no returns*)

4.7.2 Method: log

Write log message to console/file output.
Arguments:

- `msg`
  / *Condition*: optional / *Type*: str / *Default*: '' /
  Message which is written to output.
- `color`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Color style for the message.

- `indent`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Offset indent.

Returns:
(*no returns*)

4.7.3 Method: log_warning

Write warning message to console/file output.
Arguments:

- `msg`
  / *Condition*: required / *Type*: str /
  Warning message which is written to output.
- `indent`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Offset indent.

Returns:
(*no returns*)

4.7.4 Method: log_error

Write error message to console/file output.
Arguments:

- `msg`
  / *Condition*: required / *Type*: str /
  Error message which is written to output.
- `fatal_error`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  If set, tool will terminate after logging error message.
- `indent`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Offset indent.

Returns:
(*no returns*)

Chapter 5

tracker.py

5.1 Class: Status

Imported by:

```
from IssueSyncTool.tracker import Status
```

Class representing the status of issues in different tracker systems.

5.1.1 Method: normalize_issue_status

Normalize the issue status to a standard format.

Arguments:

- `tracker_type`
  / *Condition*: required / *Type*: str /
  The type of tracker (e.g., github, gitlab, jira, rtc).
- `native_status`
  / *Condition*: required / *Type*: str /
  The native status of the issue.

Returns:

- `normalized_status`
  / *Type*: str /
  The normalized status of the issue.

5.1.2 Method: get_native_status

Get the native status from the normalized status.

Arguments:

- `tracker_type`
  / *Condition*: required / *Type*: str /
  The type of tracker (e.g., github, gitlab, jira, rtc).
- `normalized_status`
  / *Condition*: required / *Type*: str /
  The normalized status of the issue.

Returns:

- `native_status`
  / *Type*: str /
  The native status of the issue.

5.2 Class: Ticket

Imported by:

```
from IssueSyncTool.tracker import Ticket
```

Normalized Ticket with required information for syncing between trackers.

5.2.1 Method: update

Update issue on tracker with following supported attributes:

- title
- assignee
- labels

Arguments:

- `kwargs`
  / *Condition*: required / *Type*: dict /
  A dictionary of attributes to update the ticket with.

5.2.2 Method: is_synced_issue

Verify whether the ticket is already synced or not.
It bases on the title of issue, it should contain destination ID information. E.g [1234] Title of already synced ticket
Returns:

- `is_synced`
  / *Type*: bool /
  Indicates if the ticket is already synced.

5.3 Class: TrackerService

Imported by:

```
from IssueSyncTool.tracker import TrackerService
```

Abstraction class of Tracker Service.

5.3.1 Method: connect

Method to connect to the tracker.

5.3.2 Method: get_ticket

Method to get a single ticket by ID from the tracker.

Arguments:

- `id`
  / *Condition*: required / *Type*: Union[str, int] /
  The ID of the ticket.

Returns:

- `ticket`
  / *Type*: Ticket /
  The ticket object.

5.3.3 Method: get_tickets

Method to get all tickets which satisfy the given condition/query.

Arguments:

- `kwargs`
  / *Condition*: optional / *Type*: dict /
  Additional keyword arguments for filtering tickets.

Returns:

- `tickets`
  / *Type*: list[Ticket] /
  A list of tickets that satisfy the given conditions.

5.3.4 Method: create_ticket

Method to create a new ticket on the tracker system.

Arguments:

- `ticket`
  / *Condition*: required / *Type*: Ticket /
  The ticket object to be created.

Returns:

- `ticket_id`
  / *Type*: str /
  The ID of the created ticket.

5.3.5 Method: exclude_issue_by_condition

Process to verify whether the given issue satisfies the exclude conditions.

Arguments:

- `issue`
  / *Condition*: required / *Type*: Issue /
  The issue object to be checked.

- `exclude_condition`
  / *Condition*: optional / *Type*: dict / *Default*: None /
  A dictionary of conditions to exclude the issue.

Returns:

- `is_excluded`
  / *Type*: bool /
  Indicates if the issue is excluded based on the given conditions.

5.3.6 Method: get_story_point_from_labels

Process to get story points from issue labels. Example of story point labels: 1 point, 2 points, ...
Arguments:

- `labels`
  / *Condition*: required / *Type*: list /
  A list of labels associated with the issue.

Returns:

- `story_points`
  / *Type*: int /
  The story points extracted from the labels.

5.3.7 Method: time_estimate_to_story_point

Convert given estimated time (in seconds) to story points.
Arguments:

- `seconds`
  / *Condition*: required / *Type*: int /
  The estimated time in seconds.

Returns:

- `story_points`
  / *Type*: int /
  The equivalent story points for the given estimated time.

5.4 Class: JiraTracker

Imported by:

```
from IssueSyncTool.tracker import JiraTracker
```

Tracker client to integrate with issues on Jira.

5.4.1 Method: connect

Connect to the Jira tracker.

Arguments:

- `project`
  / *Condition*: required / *Type*: str /
  The project name.
- `token`
  / *Condition*: required / *Type*: str /
  The access token.
- `hostname`
  / *Condition*: required / *Type*: str /
  The hostname of the Jira server.

5.4.2 Method: get_ticket

Get a ticket by its ID.

Arguments:

- `id`
  / *Condition*: required / *Type*: str /
  The ID of the ticket.

Returns:

- `ticket`
  / *Type*: Ticket /
  The ticket object.

5.4.3 Method: get_tickets

Get tickets from the Jira tracker.

Arguments:

- `kwargs`
  / *Condition*: optional / *Type*: dict /
  Additional keyword arguments for filtering tickets.

Returns:

- `tickets`
  / *Type*: list[Ticket] /
  A list of tickets that satisfy the given conditions.

5.4.4 Method: create_ticket

Create a new ticket in the Jira tracker.

Arguments:

- `project`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project name.

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for creating the ticket.

Returns:

- `ticket_id`
  / *Type*: str /
  The ID of the created ticket.

5.4.5 Method: update_ticket

Update an existing ticket in the Jira tracker.

Arguments:

- `id`
  / *Condition*: required / *Type*: str /
  The ID of the ticket to update.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for updating the ticket.

5.4.6 Method: get_story_point

Get the story points of an issue.

Arguments:

- `issue`
  / *Condition*: required / *Type*: Issue /
  The issue object.

Returns:

- `story_points`
  / *Type*: int /
  The story points of the issue.

5.4.7 Method: create_label

Jira does not require to create label before, label can be add directly in ticket.

Arguments:

- `label_name`
  / *Condition*: required / *Type*: str /
  The name of the label.
- `color`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The color of the label.
- `repository`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The repository name.

5.5 Class: GithubTracker

Imported by:

```
from IssueSyncTool.tracker import GithubTracker
```

Tracker client to integrate with issues on GitHub.

5.5.1 Method: connect

Connect to the GitHub tracker.

Arguments:

- `project`
  / *Condition*: required / *Type*: str /
  The project name.
- `repository`
  / *Condition*: required / *Type*: Union[list, str] /
  The repository name(s).
- `token`
  / *Condition*: required / *Type*: str /
  The access token.
- `hostname`
  / *Condition*: optional / *Type*: str / *Default*: "api.github.com" /
  The hostname of the GitHub server.

5.5.2 Method: get_tickets

Get tickets from the GitHub tracker.

Arguments:

- `kwargs`
  / *Condition*: optional / *Type*: dict /
  Additional keyword arguments for filtering tickets.

Returns:

- `tickets`
  / *Type*: list[Ticket] /
  A list of tickets that satisfy the given conditions.

5.5.3 Method: get_ticket

Get a ticket by its ID.

Arguments:

- `id`
  / *Condition*: required / *Type*: int /
  The ID of the ticket.

- `repository`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The repository name.

Returns:

- `ticket`
  / *Type*: Ticket /
  The ticket object.

5.5.4 Method: create_ticket

Create a new ticket in the GitHub tracker.

Arguments:

- `repository`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The repository name.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for creating the ticket.

Returns:

- `ticket_id`
  / *Type*: str /
  The ID of the created ticket.

5.5.5 Method: update_ticket

Update an existing ticket in the GitHub tracker.

Arguments:

- `id`
  / *Condition*: required / *Type*: int /
  The ID of the ticket to update.
- `repository`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The repository name.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for updating the ticket.

5.5.6 Method: create_label

Create a new label in the GitHub tracker.

Arguments:

- `label_name`
  / *Condition*: required / *Type*: str /
  The name of the label.

- `color`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The color of the label.
- `repository`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The repository name.

5.6 Class: GitlabTracker

Imported by:

```
from IssueSyncTool.tracker import GitlabTracker
```

Tracker client to integrate with issues on Gitlab.

Except, get_tickets which allow to get issues from gitlab, group and project level, other method requires project information to interact properly with inside issues.

5.6.1 Method: connect

Connect to the Gitlab tracker.

Arguments:

- `group`
  / *Condition*: required / *Type*: str /
  The group name.
- `project`
  / *Condition*: required / *Type*: Union[list, str] /
  The project name(s).
- `token`
  / *Condition*: required / *Type*: str /
  The access token.
- `hostname`
  / *Condition*: optional / *Type*: str / *Default*: "https://gitlab.com" /
  The hostname of the Gitlab server.

5.6.2 Method: get_ticket

Get a ticket by its ID.

Arguments:

- `id`
  / *Condition*: required / *Type*: int /
  The ID of the ticket.
- `project`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project name.

Returns:

- `ticket`
  / *Type*: Ticket /
  The ticket object.

5.6.3 Method: get_tickets

Get tickets from the Gitlab tracker.

Arguments:

- `kwargs`
  / *Condition*: optional / *Type*: dict /
  Additional keyword arguments for filtering tickets.

Returns:

- `tickets`
  / *Type*: list[Ticket] /
  A list of tickets that satisfy the given conditions.

5.6.4 Method: create_ticket

Create a new ticket in the Gitlab tracker.

Arguments:

- `project`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project name.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for creating the ticket.

Returns:

- `ticket_id`
  / *Type*: str /
  The ID of the created ticket.

5.6.5 Method: update_ticket

Update an existing ticket in the Gitlab tracker.

Arguments:

- `id`
  / *Condition*: required / *Type*: int /
  The ID of the ticket to update.
- `project`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project name.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for updating the ticket.

5.6.6 Method: get_story_point

Get the story points of an issue.

Arguments:

- `issue`
  / *Condition*: required / *Type*: dict /
  The issue data.

Returns:

- `story_points`
  / *Type*: int /
  The story points of the issue.

5.6.7 Method: create_label

Create a new label in the Gitlab tracker.

Arguments:

- `label_name`
  / *Condition*: required / *Type*: str /
  The name of the label.
- `color`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The color of the label.
- `project`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project name.

5.7 Class: RTCTracker

Imported by:

```
from IssueSyncTool.tracker import RTCTracker
```

Tracker client to integrate with issues on RTC (Rational Team Concert).

5.7.1 Method: connect

Connect to the RTC tracker.

Arguments:

- `project`
  / *Condition*: required / *Type*: str /
  The project name.
- `hostname`
  / *Condition*: required / *Type*: str /
  The hostname of the RTC server.

- `username`
  / *Condition*: optional / *Type*: Union[list, str] / *Default*: None /
  The username for authentication.
- `password`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The password for authentication.
- `token`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The token for authentication.
- `file_against`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The file against which to authenticate.

5.7.2 Method: get_ticket

Get a ticket by its ID.

Arguments:

- `id`
  / *Condition*: required / *Type*: Union[str, int] /
  The ID of the ticket.

Returns:

- `ticket`
  / *Type*: Ticket /
  The ticket object.

5.7.3 Method: get_tickets

Get tickets from the RTC tracker.

Arguments:

- `kwargs`
  / *Condition*: optional / *Type*: dict /
  Additional keyword arguments for filtering tickets.

Returns:

- `tickets`
  / *Type*: list[Ticket] /
  A list of tickets that satisfy the given conditions.

5.7.4 Method: get_plannedFor

Get the planned for attribute of an issue.

Arguments:

- `issue`
  / *Condition*: required / *Type*: Ticket /
  The issue object.

Returns:

- `planned_for`
  / *Type*: str /
  The planned for attribute of the issue.

5.7.5 Method: update_ticket_state

Update the state of a ticket.

Arguments:

- `issue`
  / *Condition*: required / *Type*: Ticket /
  The issue object.
- `new_state`
  / *Condition*: required / *Type*: str /
  The new state of the ticket.

5.7.6 Method: create_ticket

Create a new ticket in the RTC tracker.

Arguments:

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for creating the ticket.

Returns:

- `ticket_id`
  / *Type*: str /
  The ID of the created ticket.

5.7.7 Method: update_ticket

Update an existing ticket in the RTC tracker.

Arguments:

- `ticket_id`
  / *Condition*: required / *Type*: str /
  The ID of the ticket to update.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Additional keyword arguments for updating the ticket.

5.8 Class: Tracker

Imported by:

```
from IssueSyncTool.tracker import Tracker
```

Factory class for creating tracker instances.

5.8.1 Method: create

Create a tracker instance of the specified type.

Arguments:

- `type`
  / *Condition*: required / *Type*: str /
  The type of tracker to create.
- `args`
  / *Condition*: optional / *Type*: tuple /
  Additional positional arguments for the tracker constructor.
- `kwargs`
  / *Condition*: optional / *Type*: dict /
  Additional keyword arguments for the tracker constructor.

Returns:

- `tracker`
  / *Type*: TrackerService /
  An instance of the specified tracker type.

Raises:

- `NotImplementedError`
  If the specified tracker type is not supported.

5.8.2 Method: get_support_trackers

Get a dictionary of supported tracker types and their corresponding classes.

Returns:

- `trackers`
  / *Type*: dict /
  A dictionary where the keys are tracker types and the values are the corresponding tracker classes.

Chapter 6

user.py

6.1 Class: User

Imported by:

```
from IssueSyncTool.user import User
```

Class representing a user with a name and an ID.

6.2 Class: UserManagement

Imported by:

```
from IssueSyncTool.user import UserManagement
```

Class for managing a list of users.

6.2.1 Method: get_user

Get a user by their ID and tracker.
Arguments:

- `id`
  / *Condition*: required / *Type*: str /
  The ID of the user.
- `tracker`
  / *Condition*: required / *Type*: str /
  The tracker type (e.g., github, rtc).

Returns:

- `user`
  / *Type*: Union[User, None] /
  The user object if found, otherwise None.

Chapter 7

Appendix

About this package:

Table 7.1: Package setup

Setup parameter	Value
Name	IssueSyncTool
Version	0.0.1
Date	01.11.2024
Description	Tool to synchronize issues between trackers (Github Issue, Gitlab Issue, RTC, JIRA)
Package URL	python-issue-sync-tool
Author	Tran Duy Ngoan
Email	Ngoan.TranDuy@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 8

History

0.0.1	11/2024
Initial version	