IssueSyncTool

v. 0.3.3

Tran Duy Ngoan

06.05.2025

Contents

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.

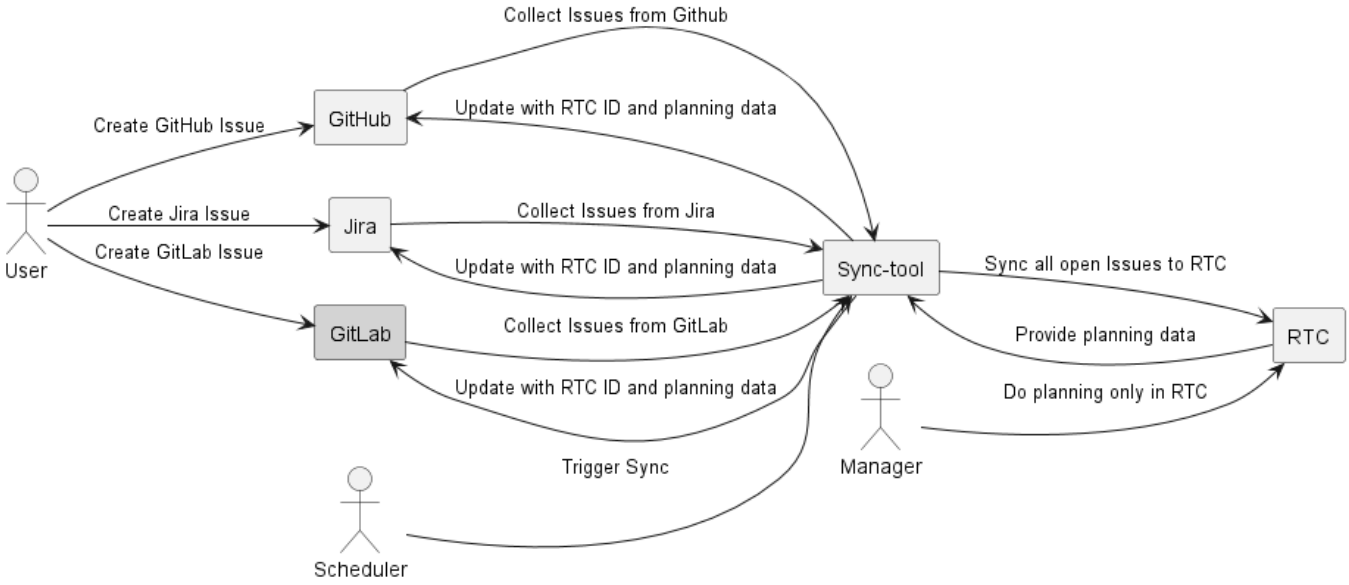

Figure 1.1: Tool's use case

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.

The tool also provides additional features to enhance the synchronization process, please refer Additional features section for more details.

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
  --nosync         If set, issues with the 'nosync' label will not be synced,
                   and any previously synced issues with this label will be closed.
  --status-only    If set, only update status of synced issue on destination tracker.
  -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.

2.3.4 Component Mapping

This configuration allows issues from specific repositories (components) to be synchronized to a destination tracker with a short name prefix in the issue title. This helps standardize and identify issues based on their originating repository in the destination tracker.

Example configuration:

```
{
   ...
   "component_mapping": {
      "python-genpackagedoc": "GenPkgDoc",
      "python-extensions-collection": "PyExtColl",
      "python-jsonpreprocessor": "JPP",
      "python-testresultdbaccess": "WebApp",
      ...
   }
   ...
}
```

Key-Value Explanation:

- Key: The name of the repository/component from which the issue originates (e.g., `python-genpackagedoc`).

- Value: The short name that will be used as a prefix in the issue title when synchronized to the destination tracker (e.g., `GenPkgDoc`).

When an issue is synchronized from a source repository to the destination tracker, the feature automatically applies the following changes:

- Looks up the repository name in the `component_mapping` configuration.
- Prepends the corresponding short name to the issue title as a prefix.

Example issue:

- Repository: `python-genpackagedoc`
- Original Issue Title: `Fix missing documentation`
- Destination Tracker Issue Title: `[GenPkgDoc] Fix missing documentation`

2.3.5 Sprint-Version Mapping Configuration

Consider a team delivering both `AIO` and `DevAtServ` in the same sprint cycle. Each product has its own development timeline and release schedule.

This configuration allows SyncTool to determine and synchronize the correct version label on the original issue, based on the sprint information found in the destination issue.

The `sprint_version_mapping` configuration defines the mapping between sprint identifiers and the specific versions of multiple software components (or products) being developed and released in parallel.

- Each key in the mapping (e.g., `PI25.2.1`, `PI25.3.1`) represents a sprint or planning increment (PI).
- Each value is a nested mapping of product names (e.g., `AIO`, `DevAtServ`) to their respective release version numbers.

Example Configuration:

```
{
   ...
   "sprint_version_mapping": {
      "PI25.2.1" : {
         "AIO" : "0.14.0",
         "DevAtServ" : "0.1.0"
      },
      "PI25.2.2" : {
         "AIO" : "0.14.1",
         "DevAtServ" : "0.1.0"
      },
      "PI25.2.4" : {
         "AIO" : "0.14.1",
         "DevAtServ" : "0.1.0"
      },
      "PI25.3.1" : {
         "AIO" : "0.14.2",
         "DevAtServ" : "0.1.0"
      }
   }
}
```

This configuration plays a critical role in environments where teams are:

- Developing **multiple software products** in parallel (e.g., `AIO`, `DevAtServ`).
- Practicing Agile or SAFe methodologies with iterative sprint-based planning.

2.4 Additional features

2.4.1 `nosync` Label

The `nosync` label is a special label that can be applied to issues in the source tracker (e.g., GitHub, Jira, Gitlab) to prevent them from being synced to the destination tracker (RTC).

When the `nosync` label is applied to an issue, the tool will not sync that issue to RTC.

If an issue was previously synced and later labeled with `nosync`, the tool will close the issue in RTC to indicate that it should no longer be tracked.

To use this feature, add the `--nosync` argument when running the tool:

```
IssueSyncTool --config your-config-file --nosync
```

2.4.2 Status-only Update

The `--status-only` argument allows you to update only the status of previously synced issues on the destination tracker (RTC).

This feature is useful when you want to update the status of issues without modifying other fields or creating new work items.

To use this feature, add the `--status-only` argument when running the tool:

```
IssueSyncTool --config your-config-file --status-only
```

2.4.3 User-defined Workflow (only for RTC destination tracker)

The tool supports user-defined workflows (state transitions) in RTC.

You can configure the tool to update the status of issues based on the workflow defined in RTC.

At least, all actions from `New` to `Done` and vice-versa should be defined in this configuration to ensure the tool can update the status of issues correctly.

Example of user-defined workflow in RTC:

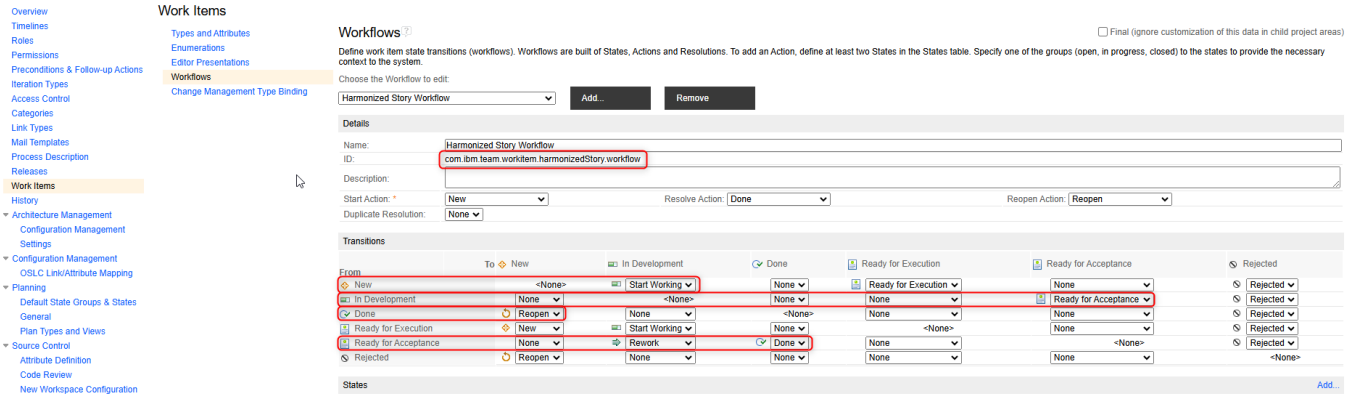

Figure 2.1: User-defined workflow in RTC

To use this feature, add the following configuration to the JSON file:

```
{
    ...
    "rtc": {
        ...
        "workflow_id" : "com.ibm.team.workitem.harmonizedStory.workflow",
        "state_transition": {
            "Start Working": [ "New", "In Development"],
            "Ready for Acceptance": ["In Development", "Ready for Acceptance"],
            "Done": ["Ready for Acceptance", "Done"],
            "Reopen": ["Done", "New"]
        }
    }
    ...
}
```

In this example:

- The `state_transitions` section defines the allowed state transitions for issues in RTC.
- For instance, an issue in the `open` state can transition to `in progress`, and so on.
- By defining the state transitions, you can ensure that issues are updated according to the workflow defined in RTC.

2.4.4 csv Output

The `--csv` argument allows you to store the sync status of issues in a CSV file named `sync_status.csv`.

This file contains information about the sync process, including the issue ID, source tracker, destination tracker, and sync status.

To use this feature, add the `--csv` argument when running the tool:

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

Sample output in `sync_status.csv`:

```
No., Ticket, Source Link, Destination ID, Stage
1, Github 2, https://github.com/ngoan1608/ntd1hc-sample-deb/issues/2, rtc , skipped
2, Github 1, https://github.com/ngoan1608/ntd1hc-sample-deb/issues/1, rtc 620036, synced
```

Possible value of status and their meaning:

- `new` : The issue is new creation in the destination tracker.
- `synced` : The issue has been successfully synced to the destination tracker.
- `not found` : The issue contains synced ID in title but was not found in the destination tracker.
- `skipped` : The issue was not existing in destination tracker and skipped during the sync process (when using `--status-only`).
- `nosync` : The issue has been labeled with `nosync` and will not be synced to the destination tracker (when using `--nosync`).
- `closed nosync` : The issue was previously synced and later labeled with `nosync` , so it has been closed in the destination tracker (when using `--nosync`).

2.4.5 Default (Configurable) Values for New RTC Work Items

When creating new work items in RTC, the tool allows you to define default values for specific fields. These values can be configured in the JSON configuration file under the `rtc` section.

Example configuration:

```
{
    ...
    "rtc": {
        ...
        "file_against": "RF-AIO",
        "planned_for": "Backlog_RFAIO",
        "project_scope": "Roadmap",
        ...
    }
    ...
}
```

Explanation of fields:

- `file_against:` Sets the default **Filed Against**.
- `planned_for:` Sets the default **Planned For**.
- `project_scope:` Specifies the default **Project Scope** for new work items.

These default values ensure that newly created work items in RTC have consistent and predefined attributes, reducing manual effort and ensuring alignment with organizational standards.

2.5 Sync Data and Attributes Mapping

2.5.1 Sync Data: Attributes

Stage	Original tracker	Direction	Destination tracker
New	Component + Title	→	Title
	URL + Description	→	Description
	Assignee	→	Assignee
	Priority	→	Priority
	Story point	→	Story point
	Labels	→	Labels
	Relationship	→	Relationship
	Title (e.g [xxx] Title)	←	ID
Sync	Component + Title	→	Title
	URL + Description	→	Description
	Relationship	→	Relationship
	Labels	→	Labels
	Sprint	←	Sprint
	Version	←	Version
	Assignee	←	Assignee (if set on Destination)
		→	Assignee (if not set on Destination)
	Story point	←	Story point (if set on Destination)
		→	Story point (if not set on Destination)
	Priority	←	Priority (is set on Destination)
		→	Priority (if not set on Destination)

2.5.2 Attributes Mapping on Trackers

Name	Github	Gitlab	JIRA	RTC
Title	Title	Title	Title	Title
URL	URL	URL	URL	URL
Assignee	Assignee	Assignee	Assignee	Owner
Component	Repository	Repository	Component	Component
Status	Labels ("in work", "ready for verifying")	Labels ("in work", "ready for verifying")	Status	Status
Story point	Labels (x pts)	Labels (x pts)	Estimate	Story point
Priority	Labels (prio x)	Labels (prio x)	Priority	Priority
Sprint	Labels (PI.*)	Labels (PI.*)	Sprint	Planned For

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 Function: escape_xml_content

Escape special XML characters.

Arguments:

- `content`
  / *Condition*: required / *Type*: str /
  The content need to be escaped.

Returns:

- / *Type*: str /
  The escaped content.

3.3 Class: RTCClient

Imported by:

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

Client for interacting with RTC (Rational Team Concert).

3.3.1 Method: get_planned_for_url

3.3.2 Method: get_project_scope_url

Get the project scope link.

Arguments:

- `project_scope`
  / *Condition*: required / *Type*: str /
  The project_scope name.

Returns:

- `project_scope_url`
  / *Type*: str /
  The link to given project_scope name.

3.3.3 Method: get_priority_link

Get the priority link.

Arguments:

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

Returns:

- `priority_identifier`
  / *Type*: str /
  The priority link for the specified value.

3.3.4 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.

3.3.5 Method: get_user_link

Get the user link for the specified user ID.

Arguments:

- `user_id`
  / *Condition*: required / *Type*: str /
  The user id on RTC.

Returns:

- / *Type*: str /
  The user URL.

3.3.6 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.

3.3.7 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.

3.3.8 Method: login

Authenticate and establish a session with RTC.

3.3.9 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.

3.3.10 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`

3.3.11 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`

3.3.12 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`

3.3.13 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.
- `priority`
  / *Condition*: optional / *Type*: int / *Default*: None /
  The priority of the work item.
- `project_id`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project ID.
- `project_scope`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project scope name.
- `type`
  / *Condition*: optional / *Type*: str / *Default*: "Story" /
  The RTC work item type.

- `state`
  / *Condition*: optional / *Type*: str / *Default*: "New" /
  The RTC work item status.
- `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: update_issue_relationship

4.2 Function: get_id_from_title

4.3 Function: get_additional_labels_of_sprint

4.4 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*)

4.5 Function: process_cli_argument

Create and configure the ArgumentParser instance, then process command-line arguments.

Returns:

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

4.6 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.

4.7 Function: process_title

Process title of the ticket with component mapping.

Arguments:

- `title`
  / *Condition*: required / *Type*: str /
  The issue title.
- `component`
  / *Condition*: optional / *Type*: str /
  The component (repository) which issue is belong to.
- `component_mapping`
  / *Condition*: optional / *Type*: dict /
  Component mappings for naming ticket title on destination tracker.

Returns:

- `title`
  / *Type*: str /
  The issue title for destination tracker.

4.8 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. The user who will be assigned to the new issue on the destination tracker.
- `component_mapping`
  / *Condition*: optional / *Type*: dict /
  Component mappings for naming ticket title on destination tracker.

Returns:

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

4.9 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.
- `component_mapping`
  / *Condition*: optional / *Type*: dict /
  Component mappings for naming ticket title on destination tracker.
- `sprint_version_mapping`
  / *Condition*: optional / *Type*: dict /
  Mappings between sprint planning and product (AIO and DevAtServ) versions.

Returns:
(*no returns*)

4.10 Function: SyncIssue

Main function to sync issues between tracking systems.

Returns:
(*no returns*)

4.11 Class: Logger

Imported by:

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

Logger class for logging messages.

4.11.1 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*)

4.11.2 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*)

4.11.3 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*)

4.11.4 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.

5.2.3 Method: get_sub_issues

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_ticket_by_condition

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

Arguments:

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

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

Returns:

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

5.3.6 Method: get_priority_from_labels

Process to get priority from issue labels. Example of priority labels: prio 1, prio 2, ...

Arguments:

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

Returns:

- `priority`
  / *Type*: int /
  The priority extracted from the labels.

5.3.7 Method: get_story_point_from_labels

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

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.

5.3.8 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_priority

Get the priority of an issue.

Arguments:

- `issue`
  / *Condition*: required / *Type*: <class 'jira.resources.Issue'> /
  The issue object.

Returns:

- `priority`
  / *Type*: int /
  The priority of the issue.

5.4.7 Method: get_priority_name_from_level

5.4.8 Method: get_story_point

Get the story points of an issue.

Arguments:

- `issue`
  / *Condition*: required / *Type*: <class 'jira.resources.Issue'> /
  The issue object.

Returns:

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

5.4.9 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.

5.4.10 Method: get_sprints

Description

Arguments:

- `board_id`
  / *Condition*: optional / *Type*: int /
  Jira Board ID to get Sprints

Returns:

/ *Type*: list /
List of Sprints which are belongs to given board ID

5.4.11 Method: get_sprint_of_issue

Get Sprint information from given issue

Arguments:

- `issue`
  / *Condition*: required / *Type*: <class 'jira.resources.Issue'> /
  Jira issue resource.

Returns:

- `sprint_name`
  / *Type*: str /
  Jira Sprint name.

5.4.12 Method: add_issues_to_sprint

Add issue to Sprint

Arguments:

- `sprint_name`
  / *Condition*: required / *Type*: str /
  Jira Sprint name.
- `issue_id`
  / *Condition*: required / *Type*: str /
  Issue id to add to given Sprint.
- `board_id`
  / *Condition*: optional / *Type*: int /
  Board ID to create new Sprint. If not given, the configured board_id of Tracker is used.

Returns:

(*no returns*)

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: get_user_id

5.6.2 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.

5.6.3 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.

5.6.4 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.

5.6.5 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.

5.6.6 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.

5.6.7 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.

5.6.8 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.
- `workflow_id`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The rtc workflow id which is current used.
- `state_transition`
  / *Condition*: optional / *Type*: dict / *Default*: None /
  The actions and according state transitions which is defined in RTC.
- `project_scope`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The project scope name to set as default for new rtc work item.
- `planned_for`
  / *Condition*: optional / *Type*: str / *Default*: None /
  The sprint name (planned_for) to set as default for new rtc work item.

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_priority

Get the priority attribute of an issue.

Arguments:

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

Returns:

- `priority`
  / *Type*: str /
  The priority attribute of the issue.

5.7.5 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.

5.7.6 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.

5.7.7 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.

5.7.8 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	0.3.3
Date	06.05.2025
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	
0.1.0	01/2025
- Add component mapping as prefix destination title in configuration json file. *- Let not assign issue as unassigned (on RTC) instead of logged in user.* *- Add new feature (optional argument –nosync) to process issue 'nosync' label.*	
0.2.0	01/2025
- Add configurations to support user-defined workflow (state transitions) in RTC. *- Add a new feature (optional argument –status-only) to update only the status of synced issues.*	
0.2.1	02/2025
- Add priority information for syncing. *- Update behavior when syncing assginee and sprint information.*	
0.2.2	02/2025
- Improve the console message when syncing fails. *- Add sprint version mapping for labeling original ticket with product's version.* *- Update the behavior to continue with the next issue instead of terminating when an error occurs during syncing or issue creation.*	
0.2.3	03/2025
Fix issue when adding label to closed ticket on JIRA	
0.2.4	03/2025
Use labels `in work` *and* `ready for verifying` *for* `In Progress` *and* `Done` *status*	
0.3.0	03/2025
Introduce new feature to sync Epic and its relationship	
0.3.1	03/2025
Update the behavior of `priority` *:* *- If the priority is not set in destination issue, sync it from original issue to destination.* *- Otherwise, sync priority back from destination to original issue.*	
0.3.2	03/2025
- Update the behavior of `story point` *and* `assignee` *to sync back the value from the destination if it is set.* *- Add new configuration* `project_scope` *for RTC tracker to set project scope when creating new work item.*	
0.3.3	05/2025

- *Add new configuration* `planned_for` *for RTC tracker to set default* `Planned For` *when creating new work item.*
- *Add feature to sync (update or create)* `Sprint` *of issue back to Jira with configured board ID.*

IssueSyncTool.pdf

Created at 06.05.2025 - 19:45:39

by GenPackageDoc v. 0.41.1