Moic

Jun 04, 2020

Contents:

1 Getting Started **3**

2 Autocompletion **5**

3 Contribute **7**
- 3.1 Prerequisites 7
- 3.2 Setup 7

Python Module Index **31**

Index **33**

Freely inspired by https://pypi.org/project/jira-cli/

Command line utility to interact with issue manager such as Jira, Gitlab, etc…

Highlights

- Modern CLI based on Click and Rich
- Context management
- Multiple tracker plugin
- Uniformed display

CHAPTER 1

Getting Started

- Install moic

```
> pip install moic
```

- Configure moic

```
> moic configure
```

- Commands

```
> moic
Usage: moic [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  config    Configure Jira cli, setting up instance, credentials etc...
  issue     Create, edit, list Jira issues
  rabbit    Print an amazing rabbit: Tribute to @fattibenji...
  resources List projects, issue_types, priorities, status
  sprint    Create, edit, list Jira Sprints
  template  List, edit templates
  version   Provide the current moic installed version
```

CHAPTER 2

Autocompletion

To activate bash autocompletion just add: * For bash

```
# In your .bashrc
eval "$(_MOIC_COMPLETE=source_bash moic)"
```

- For zsh

```
# In your .zshrc
eval "$(_MOIC_COMPLETE=source_zsh moic)"
```

CHAPTER 3

Contribute

Feel free open issues on Gitlab or propose Merge Requests

3.1 Prerequisites

This project is based on Poetry as a package manager. It allows the use of virtual environments and the lock of package versions, whether they are in your dependencies or just sub-dependencies of these dependencies.

3.2 Setup

- Create virtualenv (Optionnaly you can use pyenv which is a Python Virtualenv Manager in combination with **Poetry**)

```
poetry shell
```

- Install dependencies

```
poetry install
```

- Install pre-commit (using Pre-commit framework)

```
pre-commit install
```

  Pre-commit will check isort, black, flake8

Commit messages

This project uses semantic versioning so it is important to follow the rules of conventional commits.

3.2.1 Configure

Contexts

Moic allows you to setup multiple contexts working with different instances of issue trakers.

You can get the list of defined context with:

List and use contexts

```
> moic context list
✓ jira-cogip          : Jira Tracker of COGIP Inc
  gitlab              : Gitlab.com
```

You can switch easily from a context to another with:

```
> moic context set gitlab
Context swithed to gitlab
```

Create/delete a context

To create a new configured context, run:

```
> moic context add <plugin_name> --name my_context --description "This is my new␣
↪context"
Context created
```

Moic will ask you some basic configuration (instance, login etc...). It could store you password if needed within Keyring and will trigger the plugin specific configuration.

bq. Today only **jira** plugin is available, other plugins will be delivered soon (Gitlab, etc...)

You can delete a context with:

```
> moic context delete my_context
my_context deleted
```

Configuration file

The configuration is stored in a config file (default: `~/.moic/config.yaml`). Here is an example with Jira plugin custom configuration:

```
default:
  contexts:
  - custom_fields:
      peer: customfield_10800
      story_points: customfield_10106
    default_project: JIRA
    description: Jira Tracker of COGIP Inc
    instance: https://jira.cogip.fr
    login: bsantus
    name: jira-cogip
    plugin: jira
```

(continues on next page)

(continued from previous page)

```
    projects:
      JIRA:
        workflow:
          done:
          - '10005'
          - '10001'
          indeterminate:
          - '10002'
          - '3'
          - '10007'
          - '10009'
          new:
          - '1'
          - '10000'
  current_context: jira-cogip
```

3.2.2 Plugins

MOIC allows creation of plugins to interact with several issue tracker. Today the only plugin setup is Jira. In a near future some other will be integrated starting with Gitlab.

The creation of a plugin is quite simple.

In the future it will be simplified using plugin name convention to unleashed plugin development

Create a plugin

Plugins should be created under `moic/plugins/<plugin_name>`.

A plugin should provide several packages: * **commands**: Contains method called by Moic for each actions * **completion**: Contains autocompletion functions called by Moic * **validators**: Contains data validator functions called by Moic

Instance

First of all a plugin should expose a Instance class which herited from `MoicInstance`

```python
from moic.cli import MoicInstance, console, global_settings, settings

custom_commands = [] # use to list the custom commands which could be load by moic

class Instance(MoicInstance):

    @property
    def session(self):
        """
        Store the session object which wrap tracker access and actions.
        It's also used to check the tracker connection access.
        """
        pass

    def create_session_instance(self) -> None:
        """
```

(continues on next page)

(continued from previous page)

```
        This method is used to instanciate the session property if it doesn't exists␣
↪yet
        """
        pass

    def add_context(self, name: str, force: bool = False) -> dict:
        """
        This method should build a dict contains the tracker configuration

        Returns:
            dict: The cont
        """
        pass

    def custom_config(self, project: str, force: bool = False) -> None:
        """
        This method could be use to add more specific configuration in the context
        using self.update_config(<dict>)
        """
        pass
```

Context

A context should minimaly get the following properties: * **instance**: Tracker url * **login**: username to acces the tracker * **default_projet**: The default project id/key used by moic

Commands

For each resources and default commands, Moic will try to load a `module.method` corresponding and pass it the options/arguments provided by the user at runtime.

If a plugin, module or method doesn't exists, Moic will fallback on a warning

Example:

```
# if you call: "moic issue get" with the context using Jira plugin
# Moic will try to import
from moic.plugins.jira.commands.issue import get
```

Custom commands

If you need to add more specific method / resources than the basic one, you can just add them in the `custom_commands` list exposed by your plugin.

Moic could be able to import `moic.plugins.<plugin_name>.custom_commands`

This commands should be `click` commands and will be directly imported and exposed by Moic

Completion

Moic provides autocompletion for basic resources but you need to implement the way autocompletion retrieve results For eac

3.2.3 moic package

Subpackages

moic.cli package

Subpackages

moic.cli.commands package

Subpackages

moic.cli.commands.context package

Submodules

moic.cli.commands.context.base module

Module for base Moic configuration commands

Module contents

Module for Moic configuration commands

moic.cli.commands.issue package

Submodules

moic.cli.commands.issue.base module

Module for base Moic issue commands

Module contents

Module for base Moic issue commands

moic.cli.commands.rabbit package

Submodules

moic.cli.commands.rabbit.base module

Module for Moic fun commands

Module contents

Module which do what the f*** things

moic.cli.commands.resources package

Submodules

moic.cli.commands.resources.base module

Module for base Moic resources commands

Module contents

Module for Moic ressrouces commands

moic.cli.commands.template package

Submodules

moic.cli.commands.template.base module

Module for base Moic template commands

Module contents

Module for template commands

Submodules

moic.cli.commands.version module

Module which contains the version command for moic

Module contents

moic.cli.completion package

Submodules

moic.cli.completion.base module

Base module for completion functions It includes all function used for autocomplete click options and arguments

moic.cli.completion.base.**autocomplete_boards**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get autocompleted boards list

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available boards name

Return type list

moic.cli.completion.base.**autocomplete_comments**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get autocompleted comments list

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available comments name

Return type list

moic.cli.completion.base.**autocomplete_contexts**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get autocompleted contexts list from configuration

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available issuersautocompleted

Return type list

moic.cli.completion.base.**autocomplete_issue_types**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get issue types list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of issue types names corresponding to the incomplete input

Return type list

moic.cli.completion.base.**autocomplete_plugins**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get autocompleted plugins list

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available plugins autocompleted

Return type list

moic.cli.completion.base.**autocomplete_priorities**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get Jira priorities name list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of priorities name corresponding to the incomplete input

Return type list

moic.cli.completion.base.**autocomplete_projects**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get projects list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of project names corresponding to the incomplete input

Return type list

moic.cli.completion.base.**autocomplete_sprints**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get autocompleted sprints list

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available sprints name

Return type list

moic.cli.completion.base.**autocomplete_transitions**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get transitions available for an issue

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of translation names corresponding to the incomplete input

Return type list

moic.cli.completion.base.**autocomplete_users**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get users list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns jira.client.ResultList: List of users corresponding to the incomplete input

Return type list

Module contents

Module for click commands autocompletion

moic.cli.utils package

Submodules

moic.cli.utils.base module

Module for base Moic cli utils function

moic.cli.utils.base.**check_instance_is_up**()

Check if the Instance is accessible or not

moic.cli.utils.base.**get_plugin_autocomplete**(*autocompleter: str*)

Get the autocomplete method corresponding to the resource type within the given plugin

Parameters **autocompleter** (*str*) – Resource name to autocomplete

Returns The autocompletion function

Return type func

moic.cli.utils.base.**get_plugin_command**(*command: str*, *method: str*)

Get a command from a plugin base on the plugin name If the command or the method in the command doesn't exists, it will exit with a error message

Parameters

- **command** (*str*) – The command group used
- **method** (*str*) – The command inside the group

Returns The function which should be called

Return type func

moic.cli.utils.base.**get_plugin_custom_commands**(*plugin: str*)

Get the list of custom commands and custom group defined in the plugin

Parameters **plugin** (*str*) – Plugin name

Returns List of custom command to add to Moic

Return type list

moic.cli.utils.base.**get_plugin_instance**(*plugin: str*)
Get the instance class within the given plugin

Parameters **plugin** (*str*) – Plugin name

Returns The plugin Instance

Return type class

moic.cli.utils.base.**get_plugin_validator**(*validator: str*)
Get the validator corresponding to the resource type within the given plugin

Parameters **validator** (*str*) – The validator type name

Returns The validator function

Return type func

moic.cli.utils.base.**get_template**(*project: str*, *type: str*) → str
Get template for a given project and type

Parameters

- **project** (*str*) – A project name
- **type** (*str*) – A issue type name

Returns Path to the corresponding template

Return type str

moic.cli.utils.base.**print_comment**(*comment*, *prefix: str = ''*, *oneline: bool = False*)

moic.cli.utils.base.**print_comments**(*comments: list*, *prefix: str = ''*, *oneline: bool = False*)

moic.cli.utils.base.**print_issue**(*issue*, *prefix: str = ''*, *oneline: bool = False*, *subtasks: bool = False*, *last: bool = True*) → None
Print an issue

Parameters

- **issue** – Issue to print
- **prefix** (*str*) – Prefix to display before each issue line
- **oneline** (*bool*) – If set to true each issue will be printed on one line
- **subtasks** (*bool*) – If set to true it will display the issue's subtasks

Returns None

moic.cli.utils.base.**print_issues**(*issues: list*, *prefix: str = ''*, *oneline: bool = False*, *subtasks: bool = False*) → None
Print issue list

Parameters

- **issues** (*list*) – List of issues to print
- **prefix** (*str*) – Prefix to display before each issue line
- **oneline** (*bool*) – If set to true each issue will be printed on one line
- **subtasks** (*bool*) – If set to true it will display the issue's subtasks

Returns None

moic.cli.utils.base.**print_status**(*status*) → None
Print a Status

Parameters **status** – Status to print

Returns None

moic.cli.utils.base.**sort_issue_per_status**(*issues: list*, *project: str = None*) → list
Sort an issue liste based on the project defined workflow It will call the dedicated plugin function

Parameters

- **issues** (*list*) – The list of issues to sort
- **project** (*str*) – The project key

Returns The sorted issues list

Return type list

Module contents

Module for cli utils function

moic.cli.validators package

Submodules

moic.cli.validators.base module

Module for base Moic validator functions

moic.cli.validators.base.**validate_comment_id**(*ctx: click.core.Context*, *param: list*, *value: str*) → str
Validate a given comment id to check if it exists

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the id if it's validated

Return type str

moic.cli.validators.base.**validate_issue_key**(*ctx: click.core.Context*, *param: list*, *value: str*) → str
Validate a given issue key to check if it exists

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the key if it's validated

Return type str

moic.cli.validators.base.**validate_issue_type**(*ctx: click.core.Context*, *param: list*, *value: str*) → str

Validate a given issue type name to check if it exists

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the name if it's validated

Return type str

moic.cli.validators.base.**validate_priority**(*ctx*, *param*, *value*)

Validate a given priority name to check if it exists

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the priority name if it's validated

Return type str

moic.cli.validators.base.**validate_project_key**(*ctx: click.core.Context*, *param: list*, *value: str*) → str

Validate a given project key to check if it exists

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the key if it's validated

Return type str

moic.cli.validators.base.**validate_user**(*ctx*, *param*, *value*)

Validate a given user name to check if it exists

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the user name if it's validated

Return type str

Module contents

Module for Moic cli arguments and options validators

Submodules

moic.cli.base module

Module for Moic configuration

Configuration file is stored under CONF_DIR and contains the following architecture

default:

contexts:

- name: my_context type: <plugin> login: my_login

current_context: my_context

class moic.cli.base.**CustomSettings**
Bases: object

Class representing the current settings. It's a subtree of global_settings containing only the configuration contained into the current context

drill_down(*conf: dict*, *keys: list*) → str
Drill down into the configuration tree when the search key is composite For example : key.subkey.subsubkey

Parameters

- **conf** (*dict*) – The configuration to drill down
- **keys** (*list*) – The keys list which should be used to browse the conf

Returns The value of the key in the conf

Return type str

get(*value*, *default=None*) → str
Get a value into the Box element returned by global_settings (dynaconf.LazySetting)

Parameters **value** (*str*) – The key value to return

Returns The value within the configuration

Return type str

reload()
Force reloading configurat from {CONF_DIR}/config.yaml

class moic.cli.base.**MoicInstance**
Bases: object

MoicInstance Class represents the main class of the tool which should be herited by other Issuer Instance, such as MoicJiraInstance

add_context(*force: bool = False*) → None
Add a new context in the configuration

Parameters **force** (*bool*) – If True it doesn't check the current configuration before

create_session_instance() → None
Setup the session instance

custom_config_label = 'Would you like to run custom configuration'

delete_context(*context_name: str*) → None
Delete the given context from the contexts list. If it's the current_context, set the current context to ''

Parameters **context_name** (*str*) – The context name which should be deleted

instance = None

session
Property which retrieve the session to interact with the issuer

set_current_context(*context_name: str*) → None
Set the current configuration context to use when executing commands

Parameters **context_name** (*str*) – The context name which should be present in the contexts list

setup_home_dir() → None
Setup configuration directory. It creates the root configuration directory an empty .yaml conf file and the templates directory

Returns None

update_config(*sub_conf: dict*) → None
Update the local configuration merging in it a new dict of configuration

Parameters **sub_conf** (*dict*) – The new configuration to merge into the config.yaml file

Returns None

moic.cli.base.**merge_config**(*a: dict*, *b: dict*, *path: str = None*) → dict
Merge two config dict together

Parameters

- **a** (*dict*) – The main config dict
- **b** (*dict*) – The secondary config dict which should be merged into a
- **path** (*str*) – The path where to merde

Returns The merged dict

Return type dict

moic.cli.base.**merge_contexts**(*current_contexts: list*, *new_contexts: list*) → list
Merge two context list based on the context.name key It will update exiting contexts with new values and append non existing contexts

Parameters

- **current_contexts** (*list*) – The context list present in the configuration
- **new_contexts** (*list*) – The new context list to merge into the configuration

Returns The aggregated context list

Return type list

Module contents

Moic Cli Package

moic.plugins package

Subpackages

moic.plugins.jira package

Subpackages

moic.plugins.jira.commands package

Subpackages

moic.plugins.jira.commands.issue package

Submodules

moic.plugins.jira.commands.issue.base module

Module for base Moic issue commands

moic.plugins.jira.commands.issue.base.**add**(*summary*, *project*, *issue_type*, *priority*)
Create new issue

moic.plugins.jira.commands.issue.base.**add_subtasks**(*issue_key*, *subtask*)
Add subtasks to a Jira issue

moic.plugins.jira.commands.issue.base.**assign**(*issue_key*, *assignee*)
Assign a Jira issue

moic.plugins.jira.commands.issue.base.**comment**(*issue_key*, *comment*)
Add a comment on an issue

moic.plugins.jira.commands.issue.base.**edit_comment**(*issue_key*, *comment_id*)
Edit a comment on an issue

moic.plugins.jira.commands.issue.base.**get**(*id*, *all*, *project*, *search*, *oneline*, *subtasks*)
Get a Jira issue

moic.plugins.jira.commands.issue.base.**list_comments**(*issue_key*, *last*, *oneline*)
List comments on an issue

moic.plugins.jira.commands.issue.base.**move**(*issue_key*, *transition*)
Apply a transition on a Jira issue

moic.plugins.jira.commands.issue.base.**rank**(*issue_key*, *priority*)
Change the priority of a Jira issue

moic.plugins.jira.commands.issue.base.**set_peer**(*issue_key*, *peer*)
Define the peer user on a Jira issue

moic.plugins.jira.commands.issue.base.**show**(*issue_key*)
Show a Jira Issue description

Module contents

Module for base Moic issue commands

moic.plugins.jira.commands.resources package

Submodules

moic.plugins.jira.commands.resources.base module

Module for base Moic resources commands

moic.plugins.jira.commands.resources.base.**issue_type**()
List Jira Issue Type

moic.plugins.jira.commands.resources.base.**priorities**()
List Jira Priorities

moic.plugins.jira.commands.resources.base.**projects**()
List Jira Projects

moic.plugins.jira.commands.resources.base.**status**()
List Jira status

Module contents

Module for Moic ressrouces commands

moic.plugins.jira.commands.sprint package

Submodules

moic.plugins.jira.commands.sprint.base module

Module for base Moic sprint commands

Module contents

Module for Moic sprint commands

Module contents

moic.plugins.jira.completion package

Submodules

moic.plugins.jira.completion.base module

Base module for completion functions It includes all function used for autocomplete click options and arguments

moic.plugins.jira.completion.base.**autocomplete_boards**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list
Get autocompleted boards list

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available boards name

Return type list

moic.plugins.jira.completion.base.**autocomplete_comments**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get autocompleted comments list

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available comments id

Return type list

moic.plugins.jira.completion.base.**autocomplete_issue_types**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get Jira issue types list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of issue types names corresponding to the incomplete input

Return type list

moic.plugins.jira.completion.base.**autocomplete_priorities**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get Jira priorities name list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of priorities name corresponding to the incomplete input

Return type list

moic.plugins.jira.completion.base.**autocomplete_projects**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get Jira projects list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of project names corresponding to the incomplete input

Return type list

moic.plugins.jira.completion.base.**autocomplete_sprints**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get autocompleted sprints list

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List available sprints name

Return type list

moic.plugins.jira.completion.base.**autocomplete_transitions**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → list

Get Jira translations available for an issue

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of translation names corresponding to the incomplete input

Return type list

moic.plugins.jira.completion.base.**autocomplete_users**(*ctx: click.core.Context*, *args: list*, *incomplete: str*) → jira.client.ResultList

Get Jira users list completion

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **incomplete** (*str*) – String input to autocomplete

Returns List of users corresponding to the incomplete input

Return type jira.client.ResultList

Module contents

Module for click commands autocompletion

moic.plugins.jira.utils package

Submodules

moic.plugins.jira.utils.base module

Module for base Moic cli utils function

class moic.plugins.jira.utils.base.**Board**(*json_board: dict*)
Bases: object

Class representing a Jira Board

moic.plugins.jira.utils.base.**get_board_sprints**(*board_id: str*, *closed: bool = False*) → dict
Return le sprint list of a board

Parameters

- **board_id** (*str*) – The Jira Board ID
- **closed** (*bool*) – Indicate if we should returned only opened sprints

Returns Dict {"board_id": id, "sprints": list} of sprints

Return type dict

moic.plugins.jira.utils.base.**get_project_boards**(*project_key: str*) → list
Get the board list of a given project

This function is used waiting the 3.0.0 release of Python Jira which include it built-in

Parameters **project_Key** (*str*) – The Jira project Key used to filtered

Returns A list of boards dict

Return type list

moic.plugins.jira.utils.base.**get_sprint_issues**(*sprint_id: str*) → jira.client.ResultList
Returns list of Jira Issues linked to a given Jira Sprint

Parameters **sprint_id** (*str*) – Jira Sprint ID

Returns ResultList: List Jira Issues contained into the sprint

moic.plugins.jira.utils.base.**get_sprint_story_points**(*sprint_id: str*) → dict
Return the detailled list of story points for a given sprint Id Splitted between done points and todo points

Parameters **sprint_id** (*str*) – Jira Sprint ID

Returns {"sprint_id": id, "points": {"todo": float, "done": float}}

Return type dict

moic.plugins.jira.utils.base.**sort_issue_per_status**(*issues: list*, *project: str = None*) → list
Sort an issue liste based on the project defined workflow

Parameters

- **issues** (*list*) – The list of Jira issues to sort
- **project** (*str*) – The Jira project key

Returns The sorted Jira issues list

Return type list

moic.plugins.jira.utils.parser module

Module for a Jira Textile like to Markdown parser

class moic.plugins.jira.utils.parser.**BlockElement**(*raw: str*, *content: str = ''*, *content_type: str = None*, *content_style: str = None*, *oneline: bool = False*)

Bases: *moic.plugins.jira.utils.parser.JiraElement*

This class represent block part element such as code snippets They could be multiline or note

class moic.plugins.jira.utils.parser.**HeadElement**(*raw*, *content: str = None*, *level: int = 1*)

Bases: *moic.plugins.jira.utils.parser.JiraElement*

This class represents Header element

class moic.plugins.jira.utils.parser.**JiraDocument**(*raw: str*)

Bases: object

Class wich represent a Jira Rich text field such as an Issue Description or comment It provides methods to convert it to Markdown and prepare it for rendering through Rich

HEAD_COLORS = ['dodger_blue3', 'dodger_blue2', 'dodger_blue1', 'deep_sky_blue3', 'deep

isoneline(*line: str*)

Check if the given line is a oneline block With a pair of {quote|code|etc...} tags

Parameters **line** (*str*) – The line to check

class moic.plugins.jira.utils.parser.**JiraElement**

Bases: object

A root custom JiraElement It represent any kind of element

class moic.plugins.jira.utils.parser.**ListItemElement**(*raw: str*, *content: str = ''*, *level: int = 1*, *item_type: str = '*'*)

Bases: *moic.plugins.jira.utils.parser.JiraElement*

Class representing a list item such as: * item

class moic.plugins.jira.utils.parser.**NewLineElement**

Bases: *moic.plugins.jira.utils.parser.JiraElement*

Class representing an emtpy line

class moic.plugins.jira.utils.parser.**QuoteElement**(*raw*, *content: str = None*)

Bases: *moic.plugins.jira.utils.parser.JiraElement*

Class representing a quote (tag bq. in Jira)

class moic.plugins.jira.utils.parser.**TextElement**(*raw: str*)

Bases: object

Module contents

Module for cli utils function

moic.plugins.jira.validators package

Submodules

moic.plugins.jira.validators.base module

Module for base Moic validator functions

moic.plugins.jira.validators.base.**validate_comment_id**(*ctx: click.core.Context*, *param: list*, *value: str*) → str

Validate a given comment id to check if it exists in Jira

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the id if it's validated

Return type str

moic.plugins.jira.validators.base.**validate_issue_key**(*ctx: click.core.Context*, *param: list*, *value: str*) → str

Validate a given issue key to check if it exists in Jira

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the key if it's validated

Return type str

moic.plugins.jira.validators.base.**validate_issue_type**(*ctx: click.core.Context*, *param: list*, *value: str*) → str

Validate a given issue type name to check if it exists in Jira

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the name if it's validated

Return type str

moic.plugins.jira.validators.base.**validate_priority**(*ctx*, *param*, *value*)

Validate a given priority name to check if it exists in Jira

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the priority name if it's validated

Return type str

moic.plugins.jira.validators.base.**validate_project_key**(*ctx: click.core.Context*, *param: list*, *value: str*) → str

Validate a given project key to check if it exists in Jira

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the key if it's validated

Return type str

moic.plugins.jira.validators.base.**validate_user**(*ctx*, *param*, *value*)

Validate a given user name to check if it exists in Jira

Parameters

- **ctx** (*click.core.Context*) – click.core.Context of the given command
- **args** (*list*) – List of commands args
- **value** (*str*) – String input to validate

Returns Retrive the user name if it's validated

Return type str

Module contents

Module for Moic cli arguments and options validators

Submodules

moic.plugins.jira.api module

Jira Api custom calls module

moic.plugins.jira.api.**get_project_status**(*projectIdOrKey: str*, *url: str = None*, *login: str = None*, *password: str = None*) → dict

Get project status using a given project Key or ID

Parameters

- **projectIdOrKey** (*str*) – The Jira project ID or Key
- **url** (*str*) – The base url of the Jira instance
- **login** (*str*) – The login credential to access the API
- **password** (*str*) – The passowrd credential to access the API

Returns Dict of status existing for the given project

Return type dict

moic.plugins.jira.api.**get_project_story_status**(*projectIdOrKey: str*, *url: str = None*, *login: str = None*, *password: str = None*) → dict

Get project status using a given project Key or ID which correspond to issue type stories

Parameters

- **projectIdOrKey** (*str*) – The Jira project ID or Key
- **url** (*str*) – The base url of the Jira instance
- **login** (*str*) – The login credential to access the API
- **password** (*str*) – The passowrd credential to access the API

Returns Dict of status existing for the given project

Return type dict

moic.plugins.jira.base module

This module represents the the Jira Instance class

class moic.plugins.jira.base.**Instance**

Bases: *moic.cli.base.MoicInstance*

Instance class which allow you to access Jira's API using basic auth credentials It allows to setup several configuration such as: - Default project, custom fields, sprint workflow etc...

add_context(*name: str*, *force: bool = False*) → dict

Setup the main configuration and saved it: Instance, credentials and default project

Parameters **force** (*bool*) – Force configuration to be setup even if it exists

Returns None

create_session_instance() → None

Setup the instance if it doesn't exist yet

Returns None

custom_config(*project: str*, *force: bool = False*) → None

Configure the agile settings It configured several custom fields: * Story point custom field * Peer custom field

Parameters

- **project** (*str*) – Jira Project key which should be configured
- **force** (*bool*) – Force configuration to be setup even if it exists

Returns None

custom_config_label = 'Would you like to configure Jira Agile'

instance = None

session

Session object which represents a JIRA API session

Returns A Jira API session

Return type JIRA

moic.plugins.jira.core module

This module is used to build objects which could be used by moic.cli.utils method For example the print_issue function

class moic.plugins.jira.core.**JiraComment**(*raw: jira.resources.Comment*)
Bases: object

JiraComment Class represents a normalized issue comment based on Jira.Comment resource

class moic.plugins.jira.core.**JiraIssue**(*raw: jira.resources.Issue*)
Bases: object

JiraIssue Class represents a normalized issue based on a Jira.Issue resource

class moic.plugins.jira.core.**JiraIssueType**(*raw: jira.resources.IssueType*)
Bases: object

JiraIssueType Class represents a normalized issue type based on Jira.IssueType resource

class moic.plugins.jira.core.**JiraStatus**(*raw: jira.resources.Status*)
Bases: object

JiraStatus Class represents a normalized status based on a Jira.Status resource

Module contents

Moic Jira Package

Module contents

Submodules

moic.base module

Moic cli definition base module

moic.base.**run**()
Run the cli application

Module contents

Main Moic module

Python Module Index

m

moic, 30
moic.base, 30
moic.cli, 20
moic.cli.base, 19
moic.cli.commands, 12
moic.cli.commands.context, 11
moic.cli.commands.context.base, 11
moic.cli.commands.issue, 11
moic.cli.commands.issue.base, 11
moic.cli.commands.rabbit, 12
moic.cli.commands.rabbit.base, 11
moic.cli.commands.resources, 12
moic.cli.commands.resources.base, 12
moic.cli.commands.template, 12
moic.cli.commands.template.base, 12
moic.cli.commands.version, 12
moic.cli.completion, 15
moic.cli.completion.base, 12
moic.cli.utils, 17
moic.cli.utils.base, 15
moic.cli.validators, 18
moic.cli.validators.base, 17
moic.plugins, 30
moic.plugins.jira, 30
moic.plugins.jira.api, 28
moic.plugins.jira.base, 29
moic.plugins.jira.commands, 22
moic.plugins.jira.commands.issue, 21
moic.plugins.jira.commands.issue.base, 21
moic.plugins.jira.commands.resources, 22
moic.plugins.jira.commands.resources.base, 22
moic.plugins.jira.commands.sprint, 22
moic.plugins.jira.commands.sprint.base, 22
moic.plugins.jira.completion, 24
moic.plugins.jira.completion.base, 22
moic.plugins.jira.core, 30
moic.plugins.jira.utils, 26
moic.plugins.jira.utils.base, 25
moic.plugins.jira.utils.parser, 26
moic.plugins.jira.validators, 28
moic.plugins.jira.validators.base, 27

Index

A

add() (*in module moic.plugins.jira.commands.issue.base*), 21
add_context() (*moic.cli.base.MoicInstance method*), 19
add_context() (*moic.plugins.jira.base.Instance method*), 29
add_subtasks() (*in module moic.plugins.jira.commands.issue.base*), 21
assign() (*in module moic.plugins.jira.commands.issue.base*), 21
autocomplete_boards() (*in module moic.cli.completion.base*), 12
autocomplete_boards() (*in module moic.plugins.jira.completion.base*), 22
autocomplete_comments() (*in module moic.cli.completion.base*), 13
autocomplete_comments() (*in module moic.plugins.jira.completion.base*), 23
autocomplete_contexts() (*in module moic.cli.completion.base*), 13
autocomplete_issue_types() (*in module moic.cli.completion.base*), 13
autocomplete_issue_types() (*in module moic.plugins.jira.completion.base*), 23
autocomplete_plugins() (*in module moic.cli.completion.base*), 13
autocomplete_priorities() (*in module moic.cli.completion.base*), 14
autocomplete_priorities() (*in module moic.plugins.jira.completion.base*), 23
autocomplete_projects() (*in module moic.cli.completion.base*), 14
autocomplete_projects() (*in module moic.plugins.jira.completion.base*), 23
autocomplete_sprints() (*in module moic.cli.completion.base*), 14
autocomplete_sprints() (*in module moic.plugins.jira.completion.base*), 24
autocomplete_transitions() (*in module moic.cli.completion.base*), 14
autocomplete_transitions() (*in module moic.plugins.jira.completion.base*), 24
autocomplete_users() (*in module moic.cli.completion.base*), 15
autocomplete_users() (*in module moic.plugins.jira.completion.base*), 24

B

BlockElement (*class in moic.plugins.jira.utils.parser*), 26
Board (*class in moic.plugins.jira.utils.base*), 25

C

check_instance_is_up() (*in module moic.cli.utils.base*), 15
comment() (*in module moic.plugins.jira.commands.issue.base*), 21
create_session_instance() (*moic.cli.base.MoicInstance method*), 19
create_session_instance() (*moic.plugins.jira.base.Instance method*), 29
custom_config() (*moic.plugins.jira.base.Instance method*), 29
custom_config_label (*moic.cli.base.MoicInstance attribute*), 19
custom_config_label (*moic.plugins.jira.base.Instance attribute*), 29
CustomSettings (*class in moic.cli.base*), 19

D

delete_context() (*moic.cli.base.MoicInstance method*), 19

drill_down() (*moic.cli.base.CustomSettings method*), 19

E

edit_comment() (*in module moic.plugins.jira.commands.issue.base*), 21

G

get() (*in module moic.plugins.jira.commands.issue.base*), 21
get() (*moic.cli.base.CustomSettings method*), 19
get_board_sprints() (*in module moic.plugins.jira.utils.base*), 25
get_plugin_autocomplete() (*in module moic.cli.utils.base*), 15
get_plugin_command() (*in module moic.cli.utils.base*), 15
get_plugin_custom_commands() (*in module moic.cli.utils.base*), 15
get_plugin_instance() (*in module moic.cli.utils.base*), 16
get_plugin_validator() (*in module moic.cli.utils.base*), 16
get_project_boards() (*in module moic.plugins.jira.utils.base*), 25
get_project_status() (*in module moic.plugins.jira.api*), 28
get_project_story_status() (*in module moic.plugins.jira.api*), 28
get_sprint_issues() (*in module moic.plugins.jira.utils.base*), 25
get_sprint_story_points() (*in module moic.plugins.jira.utils.base*), 25
get_template() (*in module moic.cli.utils.base*), 16

H

HEAD_COLORS (*moic.plugins.jira.utils.parser.JiraDocument attribute*), 26
HeadElement (*class in moic.plugins.jira.utils.parser*), 26

I

Instance (*class in moic.plugins.jira.base*), 29
instance (*moic.cli.base.MoicInstance attribute*), 20
instance (*moic.plugins.jira.base.Instance attribute*), 29
isoneline() (*moic.plugins.jira.utils.parser.JiraDocument method*), 26
issue_type() (*in module moic.plugins.jira.commands.resources.base*), 22

J

JiraComment (*class in moic.plugins.jira.core*), 30
JiraDocument (*class in moic.plugins.jira.utils.parser*), 26
JiraElement (*class in moic.plugins.jira.utils.parser*), 26
JiraIssue (*class in moic.plugins.jira.core*), 30
JiraIssueType (*class in moic.plugins.jira.core*), 30
JiraStatus (*class in moic.plugins.jira.core*), 30

L

list_comments() (*in module moic.plugins.jira.commands.issue.base*), 21
ListItemElement (*class in moic.plugins.jira.utils.parser*), 26

M

merge_config() (*in module moic.cli.base*), 20
merge_contexts() (*in module moic.cli.base*), 20
moic (*module*), 30
moic.base (*module*), 30
moic.cli (*module*), 20
moic.cli.base (*module*), 19
moic.cli.commands (*module*), 12
moic.cli.commands.context (*module*), 11
moic.cli.commands.context.base (*module*), 11
moic.cli.commands.issue (*module*), 11
moic.cli.commands.issue.base (*module*), 11
moic.cli.commands.rabbit (*module*), 12
moic.cli.commands.rabbit.base (*module*), 11
moic.cli.commands.resources (*module*), 12
moic.cli.commands.resources.base (*module*), 12
moic.cli.commands.template (*module*), 12
moic.cli.commands.template.base (*module*), 12
moic.cli.commands.version (*module*), 12
moic.cli.completion (*module*), 15
moic.cli.completion.base (*module*), 12
moic.cli.utils (*module*), 17
moic.cli.utils.base (*module*), 15
moic.cli.validators (*module*), 18
moic.cli.validators.base (*module*), 17
moic.plugins (*module*), 30
moic.plugins.jira (*module*), 30
moic.plugins.jira.api (*module*), 28
moic.plugins.jira.base (*module*), 29
moic.plugins.jira.commands (*module*), 22
moic.plugins.jira.commands.issue (*module*), 21
moic.plugins.jira.commands.issue.base (*module*), 21

moic.plugins.jira.commands.resources (*module*), 22
moic.plugins.jira.commands.resources.base (*module*), 22
moic.plugins.jira.commands.sprint (*module*), 22
moic.plugins.jira.commands.sprint.base (*module*), 22
moic.plugins.jira.completion (*module*), 24
moic.plugins.jira.completion.base (*module*), 22
moic.plugins.jira.core (*module*), 30
moic.plugins.jira.utils (*module*), 26
moic.plugins.jira.utils.base (*module*), 25
moic.plugins.jira.utils.parser (*module*), 26
moic.plugins.jira.validators (*module*), 28
moic.plugins.jira.validators.base (*module*), 27
MoicInstance (*class in moic.cli.base*), 19
move() (*in module moic.plugins.jira.commands.issue.base*), 21

N

NewLineElement (*class in moic.plugins.jira.utils.parser*), 26

P

print_comment() (*in module moic.cli.utils.base*), 16
print_comments() (*in module moic.cli.utils.base*), 16
print_issue() (*in module moic.cli.utils.base*), 16
print_issues() (*in module moic.cli.utils.base*), 16
print_status() (*in module moic.cli.utils.base*), 16
priorities() (*in module moic.plugins.jira.commands.resources.base*), 22
projects() (*in module moic.plugins.jira.commands.resources.base*), 22

Q

QuoteElement (*class in moic.plugins.jira.utils.parser*), 26

R

rank() (*in module moic.plugins.jira.commands.issue.base*), 21
reload() (*moic.cli.base.CustomSettings method*), 19
run() (*in module moic.base*), 30

S

session (*moic.cli.base.MoicInstance attribute*), 20
session (*moic.plugins.jira.base.Instance attribute*), 29
set_current_context() (*moic.cli.base.MoicInstance method*), 20
set_peer() (*in module moic.plugins.jira.commands.issue.base*), 21
setup_home_dir() (*moic.cli.base.MoicInstance method*), 20
show() (*in module moic.plugins.jira.commands.issue.base*), 21
sort_issue_per_status() (*in module moic.cli.utils.base*), 17
sort_issue_per_status() (*in module moic.plugins.jira.utils.base*), 25
status() (*in module moic.plugins.jira.commands.resources.base*), 22

T

TextElement (*class in moic.plugins.jira.utils.parser*), 26

U

update_config() (*moic.cli.base.MoicInstance method*), 20

V

validate_comment_id() (*in module moic.cli.validators.base*), 17
validate_comment_id() (*in module moic.plugins.jira.validators.base*), 27
validate_issue_key() (*in module moic.cli.validators.base*), 17
validate_issue_key() (*in module moic.plugins.jira.validators.base*), 27
validate_issue_type() (*in module moic.cli.validators.base*), 17
validate_issue_type() (*in module moic.plugins.jira.validators.base*), 27
validate_priority() (*in module moic.cli.validators.base*), 18
validate_priority() (*in module moic.plugins.jira.validators.base*), 27
validate_project_key() (*in module moic.cli.validators.base*), 18
validate_project_key() (*in module moic.plugins.jira.validators.base*), 28
validate_user() (*in module moic.cli.validators.base*), 18
validate_user() (*in module moic.plugins.jira.validators.base*), 28