RoboTA

Release 2.2.2

Peter Crowther

Jun 20, 2022

CONTENTS

RoboTA (Robot Teaching Assistant) is a Python module to provide a framework for the assessment of software engineering. The focus of RoboTA is the assessment of student software engineering courswork, though it has a wider scope in the assessment of general good practice in software engineering.

The robota-core package collects information about a project from a number of sources, git repositories, issue trackers, ci-servers. It is designed to be provider agnostic, for example repository data can come from GitLab or GitHub.

There then a number of other RoboTA packages that use this information to assess project quality. robota-common-errors identifies common errors in software engineering workflows. robota-progress provides a simple progress dashboard for a project. robota-marking provides a framework for the assessment of student coursework.

RoboTA was developed in the Computer Science department at the University of Manchester. From 2018 to March 2021, development of RoboTA was funded by the Institute of Coding.

CHAPTER

ONE

INSTALLATION

To install as a Python module, type

```
python -m pip install robota-core
```

from the root directory. For developers, you should install in linked .egg mode using

```
python -m pip install robota-core -e
```

If you are using a Python virtual environment, you should activate this first before using the above commands.

CHAPTER

TWO

ROBOTA CONFIG

RoboTA requires access to a number of data sources to collect data to operate on. Details of these data sources and information required to connect to them is provided in the robota config yaml file. Documentation on the config file can be found in the *RoboTA config* section of the documentation. robota config template files are provided with the robota-common-errors, robota-progress and robota-marking packages.

2.1 API Reference

2.1.1 RoboTA config

RoboTA reads in various types of data from different sources. This could be data about software engineering objects such as git commits or it could be about the human elements of software engineering such as interactions with an issue board or bug tracker.

These data types and their sources are specified in the robota-config file.

Data types

Each data type provides a different type of information to RoboTA that is used somewhere in the code. Depending on what parts of RoboTA are being used, not all the data types may be required.

Every data type requires a 'data_source' key which says where the data comes from. Some data types have additional keys which are required to specify details of the data type.

marking_config

The location of the set up files for RoboTA marking.

Valid data sources:

- local_path
- gitlab

Required keys:

- course_config_file: The path of the Course config file
- mark_scheme_file: The path of the file that specifies the mark scheme
- build_config_file: The path to the file that has information about the build config

All of these keys may specify sub-folder(s) in the git repository, e.g.
course_config_file: config_files/course/course_config.yaml

issues

The location of issues. Issues are used in marking students planning and teamwork.

Valid sources:

- gitlab
- github

ci

A CI server hosting tests assessing student code.

Valid sources:

- jenkins

repository

A source of information about commits, tags, events, branches and files in the student work repository.

Valid sources:

- gitlab
- github
- local_repository

remote_provider

A cloud provider that hosts git repositories. Provides info about pull/merge requests and team members

Valid sources:

- gitlab
- github

attendance

Information about student attendance at a class, workshop or event

Valid sources:

- benchmark

student_details

Information about student names and usernames

Valid sources:

- gitlab
- local_path

student_emails

The relationship between student usernames and email addresses

Valid sources:

- gitlab
- local_path

required keys:

- name_list_file - The path of the file containing the mapping between names and email addresses

ta_marks

Manually assigned marks that can be used to override RoboTA marks in the marking report

Valid sources:

- gitlab
- local_path

required keys:

- ta_marks_file - The path of the file containing the TA marks.

Data Sources

Each data source specified in a data type should correspond to a data source in the data sources section. For flexibility the data sources are specified by name which means that each data source could be used for multiple data types.

Each data source has a type. Each data source type is treated differently by the code.

The different data sources are:

- local_path
- gitlab
- github
- local_repository
- jenkins
- benchmark

Below are specified the keys which are required for each data source type.

local_path

Download files from a local directory on the machine on which RoboTA is running.

required sub-values:

- path - The path to look in for the file(s).

gitlab

Connect to a remote gitlab instance to retrieve information or files.

required sub-values:

- url: The url of the gitlab instance
- project: The name of the gitlab project to load
- token: An authentication token to connect to the gitlab instance

optional sub-values:

- branch: Which git branch to assess - defaults to 'master'

github

Connect to a remote GitHub instance to retrieve information or files.

required sub-values:

- url: The url of the GitHub instance
- project: The name of the GitHub project to load
- token: An authentication token to connect to the GitHub instance

optional sub-values:

- branch: Which git branch to assess - defaults to 'master'

local_repository

Connect to a repository on the local machine.

required sub-values:

- path: The path of the git repository.

optional sub-values:

- branch: Which branch to consider - defaults to 'master'

jenkins

Connect to a remote Jenkins instance to retrieve job information

required sub-values:

- url: The url of the Jenkins instance
- username: Username used for authentication
- token: Token used for authentication
- project_name: The name of the project containing the tests
- folder_name: The name of the folder in the project containing the tests

benchmark

A University of Manchester service that has information about students

required sub-values:

- url: The URL of the benchmark instance
- token: Token used for authentication

Note that a connection to benchmark requires either being on campus or use of the UoM VPN.

Example Config

```
# This is an example robota config file.
# This file is used it to store RoboTA config variables and credentials locally.
# You should NOT commit any credentials to git.

# The 'data_types' section specifies where the data to run RoboTA comes from. The keys are
the data type and are fixed as they are specified in the code.
# The data source key is mandatory for each data type. Other key: value pairs are passed
# into the code to be used for configuration of that data type.

data_types:
    issues:
        data_source: github_repo
    repository:
        data_source: github_repo
    remote_provider:
        data_source: github_repo

# Details of data sources. The name of each data source corresponds to those specified␣
↪in the data_types section above.
# Keys and values are specific to the data source.

data_sources:
    github_repo:
        type: github
        url: www.github.com
        project: merrygoat/chi4
```

(continues on next page)

(continued from previous page)

```
        token: xxx-xxx-xxx

    local_repository:
        type: local_path
        directory: C:/robota/chi_4
```

In this case repository could probably be set to the data source: `github_repo`, but it might be useful to set it to `local_repository` if the repository was already synced locally and was large. Operating on large repositories locally is likely to be more efficient in most cases than querying them through the API.

Variable Substitution

To improve automation, named keys in the config file can be specified which are replaced by values at run time. Strings to be substituted should be enclosed in curly brackets.

The second argument of the *robota_core.config_readers.get_robota_config()* method is a dictionary of substitutions. The keys are the variable to replace and the values are the values to substitute in.

An example robota config might look like:

```
data_types:
    repository:
        data_source: student_repo

data_sources:
    student_repo:
        type: github
        url: www.gitlab.com
        project: UoMProgramming/first_year/Team{team_number}
        token: xxx-xxx-xxx
```

To loop assessment over many teams you could read in the config in a loop using the *robota_core.config_readers.get_robota_config()* method. On the first loop the *substitution_vars* parameter would be {"team_number": 01}, on the second loop, {"team_number": 02} etc.

2.1.2 robota_core

robota_core package

Submodules

robota_core.attendance module

A module to collect statistics on student attendance.

exception robota_core.attendance.**AttendanceError**

Bases: Exception

An error in collecting attendance data.

class robota_core.attendance.**StudentAttendance**(*robota_config: dict*, *mock: bool = False*)

Bases: object

The student attendance class collects data from an external API about student attendance.

Parameters

- **robota_config** – A dictionary of information about data sources read from the robota config file.
- **mock** – If True, return mock data instead of getting real data from the data source.

get_student_attendance(*student_id: str*) → int

For an individual student, get their attendance from the list of all attendances.

Parameters
student_id – The university ID name of the student to get attendance of.

Return student_attendance
The number of sessions attended in the current year.

robota_core.ci module

Module that defines interactions with a Continuous integration server in order to get build information.

class robota_core.ci.**Build**(*jenkins_build*)

Bases: object

A Build is the result of executing a CI job.

Variables

- **number** – The number of the build.
- **result** – The result of the build
- **timestamp** – The time at which the build started.
- **commit_id** – The ID of the git commit the build was run on.
- **branch_name** – The git branch of the commit the build was run on.
- **link** – A HTML string linking to the web-page that displays the build on Jenkins.
- **instruction_coverage** – A code coverage result from JaCoCo.

build_from_jenkins(*jenkins_build*)

Create a Robota Build object from a Jenkins build object.

class robota_core.ci.**BuildResult**(*value*)

Bases: Enum

Represents the result of a Jenkins build.

Aborted = 4

Failure = 3

Gitlab_Timeout = 6

Not_Built = 5

Success = 1

Unstable = 2

class robota_core.ci.**CIServer**

Bases: ABC

A CIServer is a service from which test results are fetched. All of these are abstract methods implemented by subclasses.

abstract get_job_by_name(*job_name: str*) → Optional[*Job*]

Get a job by its name. Return None if job not found.

abstract get_jobs_by_folder(*folder_name: str*) → List[*Job*]

Get all jobs located in a particular folder.

abstract get_package_coverage(*job_path: str*, *package_name: str*) → Union[None, float]

Get the percentage test coverage for a particular package.

abstract get_tests(*job_path: str*) → Union[None, List[*Test*]]

Get all Tests that were run for a job.

class robota_core.ci.**JenkinsCIServer**(*ci_source: dict*)

Bases: *CIServer*

With Jenkins it is possible to download all of the jobs from a whole project at once. This is much quicker than getting each job one by one as the API requests are slow. For this reason the JenkinsCIServer class downloads all jobs from a project and then helper methods get jobs from the local cache.

Connects to Jenkins and downloads all jobs. If the jobs are heavily nested in folders, it may be necessary to increase the depth parameter to iteratively fetch the lower level jobs.

Parameters

ci_source – A dictionary of config info for setting up the JenkinsCIServer.

get_job_by_name(*job_name: str*) → Optional[*Job*]

Get a job by its name. Return None if job not found.

get_jobs_by_folder(*folder_name: str*) → List[*Job*]

Get all jobs that were located in a particular folder.

get_package_coverage(*job_path: str*, *package_name: str*) → Union[None, float]

Get the percentage test coverage for a particular package.

Parameters

- **job_path** – The tag or job name to query.
- **package_name** – The name of the package to get coverage for.

get_tests(*job_path: str*) → Union[None, List[*Test*]]

Get Tests for a job - this is a separate API request to the main job info.

class robota_core.ci.**Job**(*job_data*, *project_root*)

Bases: object

A job is a series of CI checks. Each time a job is executed it stores the result in a build.

get_build_by_commit_id(*commit_id*) → Optional[*Build*]

Get a job triggered by commit_id

get_build_by_number(*number*) → Optional[*Build*]

Get build of this job by number, where 1 is the chronologically earliest build of a job. If build is not found, returns None.

get_builds() → List[*Build*]

Get all builds of a job.

get_first_build(*start: datetime*, *end: datetime*) → Union[None, *Build*]

Return the first (oldest) build in the time window.

get_first_successful_build(*start: datetime*, *end: datetime*) → Union[None, *Build*]

Return the first (oldest) successful build in the time window.

get_last_build(*start: datetime*, *end: datetime*) → Union[None, *Build*]

Get most recent job build status between start and end.

Parameters

- **start** – Build must occur after this time
- **end** – Build must occur before this time

Returns

Last build in time window, None if no job in time window.

get_last_completed_build() → Optional[*Build*]

"Get the last completed build of a job.

job_from_jenkins(*jenkins_job: dict*, *project_root: str*)

Create a Robota Job object from a Jenkins Job object.

class robota_core.ci.**Test**(*suite: dict*, *case: dict*)

Bases: object

A representation of the result of a Test.

Variables

- **name** – The name of the test.
- **result** – The result of the test, PASSED or FAILED.
- **time** – The time that the test ran.
- **branch** – The branch of commit the test was run upon. This is not populated on object creation.

robota_core.ci.**new_ci_server**(*robota_config: dict*) → Union[None, CIType]

Factory method for creating CIServers

robota_core.commit module

Objects and for describing and processing Git commits.

class robota_core.commit.**Commit**(*commit*, *commit_source: str*, *project_url: Optional[str] = None*)

Bases: object

An abstract object representing a git commit.

Variables

- **created_at** – (datetime) Commit creation time.

- **id** – Commit id.
- **parent_ids** – (List[string]) The ids of one or more commit parents.
- **raw_message** – (str) The original commit message
- **message** – (str) The commit message cleaned for HTML display.
- **merge_commit** – (bool) Whether this commit is a merge commit.

commit_from_local(*commit: Commit*)

get_comments() → List[str]

Return the text of all comments to this commit.

class robota_core.commit.**CommitCache**(*start: datetime*, *end: datetime*, *branch: str*, *commits: List[*Commit*]*)

Bases: object

A cache of Commit objects from a specific date range and branch.

class robota_core.commit.**CommitComment**(*comment_data*, *source: str*)

Bases: object

A comment made on a commit.

class robota_core.commit.**Tag**(*tag_data*, *source: str*)

Bases: object

A tag is a named pointer to a git commit.

Variables

- **name** – The name of the tag.
- **commit_id** – The id of the commit that the tag points to.

tag_from_dict(*tag_data: dict*)

tag_from_github(*tag_data: Tag*)

tag_from_gitlab(*tag_data: ProjectTag*)

tag_from_local(*tag_data: TagReference*)

robota_core.commit.**get_merge_commit**(*feature_tip:* Commit, *master_commits: List[*Commit*]*) → Optional[*Commit*]

Get merge commit ID for the branch "branch_title". Given the id of the commit at the tip of a feature branch, find where it merges into the master branch by going through the commits ids on the master branch and looking at their parents.

Parameters

- **feature_tip** – The Commit at the tip of the feature branch.
- **master_commits** – Commits of master branch, ordered by date, most recent first.

Return merge_commit

The id of the merge commit if branch was merged else returns None.

robota_core.commit.**get_tags_at_date**(*date: datetime*, *tags: List[*Tag*]*, *events: List[*Event*]*) → List[*Tag*]

robota_core.config_readers module

exception robota_core.config_readers.**RobotaConfigLoadError**

Bases: Exception

The error raised when there is a problem loading the configuration

exception robota_core.config_readers.**RobotaConfigParseError**

Bases: Exception

The error raised when there is a problem parsing the configuration

robota_core.config_readers.**get_config**(*file_names: Union[str, List[str]]*, *data_source: dict*) → list

The base method of the class. Calls different methods to get the config depending on the config type.

Parameters

- **file_names** – The names of one or more config files to open.
- **data_source** – Information about the source of the config data. The 'type' key specifies the source of the data and other keys are extra information about the source like url or API token.

Returns

a list of parsed file contents, one list element for each file specified in *file_names*. If a file is not found, the corresponding list element is set to None.

robota_core.config_readers.**get_data_source_info**(*robota_config: dict*, *key: str*) → Optional[dict]

Get the information about the data source specified by 'key' from the robota_config dictionary.

robota_core.config_readers.**get_gitlab_config**(*config_variables: dict*) → Tuple[Path, str]

Get config from a Gitlab repository by logging in using an access token and downloading the files from the repository.

Parameters

config_variables – required keys/value pairs are: gitlab_url: The URL of the gitlab repository gitlab_project: The full project name of the project containing the config files. gitlab_token: The gitlab access token.

Returns

the temporary directory containing the config files.

robota_core.config_readers.**get_robota_config**(*config_path: str*, *substitution_vars: dict*) → dict

The robota config specifies the source for each data type used by RoboTA. The RoboTA config is always stored locally since it contains API tokens. :param config_path: The path of the robota config file to read. :param substitution_vars: An optional dictionary of values to substitute into the config file.

robota_core.config_readers.**parse_config**(*config_path: Path*) → dict

Parses a config file to extract the configuration variables from it.

Parameters

config_path – the full file path to the config file.

Returns

the config variables read from the file. Return type depends on the file type.

robota_core.config_readers.**process_yaml**(*yaml_content: dict*) → dict

Do custom string substitution to the dictionary produced from reading a YAML file. This is not part of the core YAML spec. This function replaces instances of ${key_name} in strings nested as values in dicts or lists with the value of the key "key_name" if "key_name" occurs in the root of the dictionary.

robota_core.config_readers.**read_csv_file**(*csv_path: Union[str, Path]*) → dict

Parse a two column csv file. Return dict with first column as keys and second column as values.

robota_core.config_readers.**read_yaml_file**(*config_location: Path*) → dict

Read a YAML file into a dictionary

Parameters
config_location – the path of the config file

Returns
Key-value pairs from the config file

robota_core.config_readers.**rmtree_error**(*func*, *path*, *_*)

Error handler for `shutil.rmtree`.

If the error is due to an access error (read only file) it attempts to add write permission and then retries.

robota_core.config_readers.**substitute_dict**(*input_value: object*, *root_keys: dict*) → object

If *input_value* is a list or dict, recurse into it trying to find strings. If *input_value* is a string then substitute any variables that occur as keys in *root_keys* with the values in *root_keys*. Variables to be substituted are indicated by a bash like syntax, e.g. ${variable_name}.

robota_core.config_readers.**substitute_keys**(*robota_config: dict*, *command_line_args: dict*) → dict

Go through all of the data sources replacing any curly bracketed strings by named variables provided to roboTA as command line arguments. This allows substitution of things like a team name or exercise number into the robota config.

Parameters

- **robota_config** – The dictionary of data sources loaded from robota-config.yaml.
- **command_line_args** – Command line arguments given to RoboTA.

robota_core.data_server module

class robota_core.data_server.**DataServer**(*robota_config: dict*, *start: datetime*, *end: datetime*)

Bases: `object`

A container for data sources.

get_valid_sources() → List[str]

Check whether each of the data sources has been successfully initialised.

robota_core.github_tools module

class robota_core.github_tools.**GithubServer**(*setup: dict*)

Bases: `object`

A connection to GitHub. Contains methods for interfacing with the API.

Initialise the connection to the server, getting credentials from the credentials file.

Parameters
setup – dictionary containing GitHub url and authentication token.

open_github_repo(*project_path: str*) → Repository

Open a GitLab repo.

Parameters
project_path – The path of the project to open. Includes namespace.

Returns
A GitLab project object.

robota_core.gitlab_tools module

General methods for interfacing with GitLab via the python-Gitlab library.

class robota_core.gitlab_tools.**GitlabGroup**(*gitlab_connection: Gitlab*, *group_name: str*)

Bases: object

A group is distinct from a project, a group may contain many projects. Projects contained in a group inherit the members of the containing project.

get_group_members() → List[str]

Get a list of members in a group.

Return member_list
Names of members of group.

class robota_core.gitlab_tools.**GitlabServer**(*url: str*, *token: str*)

Bases: object

A connection to the Gitlab server. Contains methods for interfacing with the API. This is held distinct from the Repository object because it can also be used to interface with an Issue server.

Initialise the connection to the server, getting credentials from the credentials file.

Parameters

- **url** – url of GitLab server
- **token** – Authentication token for gitlab server.

open_gitlab_group(*group_name: str*) → *GitlabGroup*

open_gitlab_project(*project_path: str*) → Project

Open a GitLab project.

Parameters
project_path – The path of the project to open. Includes namespace.

Returns
A GitLab project object.

robota_core.issue module

Objects and for describing and processing Git Issues.

class robota_core.issue.**GitHubIssueServer**(*issue_server_source: dict*)

Bases: *IssueServer*

class robota_core.issue.**GitLabIssueServer**(*issue_source: dict*)

Bases: *IssueServer*

An IssueServer with GitLab as the server.

class robota_core.issue.**Issue**(*issue*, *issue_source: str*, *get_comments=True*)

Bases: object

An Issue

Variables

- **created_at** – (datetime) The time at which the issue was created.
- **assignee** – (string) The person to whom the issue was assigned.
- **closed_at** – (datetime) The time at which the issue was closed.
- **closed_by** – (string) The person who closed the issue.
- **time_stats** – (dict) Estimates and reported time taken to work on the issue.
- **due_date** – (datetime) The time at which issue is due to be completed.
- **title** – (string) The title of the issue.
- **comments** – (List[Comment]) A list of Comments associated with the Issue.
- **state** – (string) Whether the issue is open or closed.
- **milestone** – (string) Any milestone the issue is associated with.
- **url** – (string) A link to the Issue on GitLab.

get_assignee() → Optional[str]

Return name of issue assignee

:return If issue has an assignee, returns their name else returns None.

get_assignment_date() → Optional[datetime]

Get assignment date for an issue. First checks comments for assignment date and if none is found, returns the issue creation date. If there is more than one assignment date, this method will always return the most recent.

Returns

The date at which the issue was assigned.

get_comment_timestamp(*key_phrase: str*, *earliest=False*) → Optional[datetime]

Search for a phrase in the comments of an issue If the phrase exists, return creation time of the comment.

Parameters

- **key_phrase** – a phrase to search for in a comment on the issue.
- **earliest** – If True, return the earliest comment matching key_phrase, else return most recent comment matching key_phrase.

Returns
If phrase is present in a comment, return the the time of the comment, else return None

`get_date_of_time_spent_record`(*key_phrase: str*) → Union[datetime, str]

Determine whether a time spent category has been recorded. The key phrase should appear in a comment to indicate what the time has been spent on.

Parameters
key_phrase – Phrase to search for, which should have a time record associated with it

Returns
Last edited time of comment recording time spent

`get_recorded_team_member`(*key_phrase: str*) → Union[None, List[str]]

Report whether a team member has been recorded using a key phrase for issue. Key phrase should appear at the start of a comment to indicate assignment of sub-team member, code reviewer (etc).

Parameters
key_phrase – Phrase to search for

Return team_member_recorded
Str

`get_status`(*deadline: datetime*)

Get current status of issue if deadline hasn't passed, otherwise get last status of issue before the deadline, and save in the issue.state attribute so that it is only calculated once.

Parameters
deadline –

Returns

`get_time_estimate`() → timedelta

Gets estimate of time it will take to close issue.

`get_time_estimate_date`() → Optional[datetime]

Gets the date a time estimate was added to an issue. This only works for issues made after 05/02/19 as this was a feature added in Gitlab 11.4.

Returns
Date of the first time estimate, None if no time estimate was found.

`is_assignee_contributing`(*team*) → Union[bool, str]

Determine whether the Student assigned to work on an Issue is contributing to the exercise.

`class robota_core.issue.IssueCache`(*start: Optional[datetime] = None*, *end: Optional[datetime] = None*, *get_comments=True*, *milestone=None*)

Bases: `object`

A cache of Issue objects from a specific date range.

`add_issue`(*issue:* Issue)

Add an Issue to an IssueCache.

`class robota_core.issue.IssueComment`(*comment*, *source: str*)

Bases: `object`

A comment is a textual field attached to an Issue

Variables

- **text** – (string) The content of the comment message.

- **created_at** – (datetime) The time a comment was made.
- **updated_at** – (datetime) The most recent time the content of a comment was updated.

class robota_core.issue.**IssueServer**

Bases: object

An IssueServer is a service from which Issues are extracted.

get_issues(*start: datetime = datetime.datetime(1970, 1, 1, 0, 0, 1)*, *end: datetime = datetime.datetime(2022, 6, 20, 12, 16, 24, 945157)*, *get_comments: bool = True*) → List[*Issue*]

Get issues from the issue provider between the start date and end date.

get_issues_by_milestone(*milestone_name: str*) → Optional[List[*Issue*]]

Get a list of issues associated with a milestone.

robota_core.issue.**get_issue_by_title**(*issues: List[*Issue*]*, *title: str*) → Optional[*Issue*]

If issue with 'title' exists in 'issues', return the issue, else return None.

Parameters

- **issues** – A list of Issue objects.
- **title** – An issue title

Returns

Issue with title == title, else None.

robota_core.issue.**new_issue_server**(*robota_config: dict*) → Union[None, *IssueServer*]

A factory method for IssueServers.

robota_core.logic module

robota_core.logic.**are_list_items_in_other_list**(*reference_list: List*, *query_list: List*) → List[bool]

Check whether items in query_list exist in correct_list.

Parameters

- **reference_list** – The reference list of items
- **query_list** – The items to check - does this list contain items in reference_list?

Return items_present

Whether items in the correct list are in query_list (bool)

```
>>> are_list_items_in_other_list([1, 2, 3], [3, 1, 1])
[True, False, True]
```

robota_core.logic.**are_lists_equal**(*list_1: list*, *list_2: list*) → List[bool]

Elementwise comparison of lists. Return list of booleans, one for each element in the input lists, True if element N in list 1 is equal to element N in list 2, else False.

robota_core.logic.**date_is_before**(*date1: datetime*, *date2: datetime*) → bool

If date1 and date2 are provided and date1 is before date2 return True, else return False.

robota_core.logic.**find_commit_in_list**(*commit_id: str*, *commits: List[*Commit*]*) → Union[None, *Commit*]

Find a Commit in a list of Commits by its ID.

Parameters

- **commit_id** – The id of the commit to find.
- **commits** – The list of Commits to search.

Returns
Commit if found, else None.

robota_core.logic.**find_feature_parent**(*feature_commit: Commit*, *base_commits: List[Commit]*) → bool

Determine whether the provided feature commit has a commit in the base branch with a common parent.

Parameters

- **feature_commit** – The feature commit being checked.
- **base_commits** – A list of the base commits, most recent first.

Returns
True if feature_commit has a common parent with a commit in the base branch else False.

robota_core.logic.**fixup_first_feature_commit**(*feature_branch_commits: List[Commit]*, *initial_guess_of_first_commit: Commit*, *merge_commits: List[Commit]*)

Fix-up function to look for merge commits on master branch before the tip of the feature branch. Any commits up to and including a merge commit in the history of a feature branch cannot be the first commit on the feature branch.

Parameters
feature_branch_commits –

Returns

robota_core.logic.**fraction_of_lists_equal**(*list_1: list*, *list_2: list*) → float

Returns the fraction of list elements are equal when compared elementwise.

robota_core.logic.**get_first_feature_commit**(*base_commits: List[Commit]*, *feature_commits: List[Commit]*) → Optional[*Commit*]

Get the first commit on a feature branch. Determine first commit by looking for branching points, described by the parent commits. All parameters are lists of commit IDs ordered from newest to oldest

Parameters

- **base_commits** – commits from base branch (usually master)
- **feature_commits** – commits from feature branch

Return first_feature_commit
commit ID of first commit on feature

robota_core.logic.**get_value_from_list_of_dicts**(*list_of_dicts: List[dict]*, *search_key: str*, *search_value: int*, *return_key: str*)

Given a list of dictionaries, identify the required dictionary which contains the *search_key*: *search_value* pair. Return the value in that dictionary associated with *return_key*.

robota_core.logic.**is_date_before_other_dates**(*query_date: datetime*, *deadline: datetime*, *key_date: datetime*) → bool

Determine whether an action was before the deadline and another key date.

Parameters

- **query_date** – The date of the action in question, e.g. when was an issue assigned, time estimate set, or due date set.
- **deadline** – The deadline of the action

- **key_date** – The query date should be before this significant date, as well as the deadline e.g. branch creation date

Returns

True if issue query_date was before deadline and key_date else False.

robota_core.logic.**logical_and_lists**(*list1: List[bool]*, *list2: List[bool]*) → List[bool]

For two lists of booleans of length N, return a list of length N where output[i] is True if list1[i] is True and list2[i] is True, else output[i] is False.

robota_core.merge_request module

class robota_core.merge_request.**MergeRequest**(*merge_request*, *source: str*)

Bases: object

A Merge Request

class robota_core.merge_request.**MergeRequestCache**(*start: datetime*, *end: datetime*, *merge_requests: List[*MergeRequest*]*)

Bases: object

A cache of MergeRequest objects from a specific date range.

add_merge_request(*merge_request:* MergeRequest)

Add a MergeRequest to a MergeRequestCache.

class robota_core.merge_request.**MergeRequestComment**(*comment*, *source: str*)

Bases: object

Comments on a merge request

robota_core.remote_provider module

class robota_core.remote_provider.**GithubRemoteProvider**(*provider_source: dict*)

Bases: *RemoteProvider*

get_members() → Dict[str, str]

This method returns names and usernames of repo collaborators since github doesn't have the idea of members in the same way as gitlab.

class robota_core.remote_provider.**GitlabRemoteProvider**(*provider_source: dict*)

Bases: *RemoteProvider*

get_members() → Dict[str, str]

Get a dictionary of names and corresponding usernames of members of this repository.

class robota_core.remote_provider.**RemoteProvider**

Bases: object

A remote provider is a cloud provider that a git repository can be synchronised to. Remote providers have some features that a basic git Repository does not including merge requests and teams.

abstract get_members() → Dict[str, str]

Get a dictionary of names and corresponding usernames of members of this repository.

get_merge_requests(*start: datetime = datetime.datetime(1970, 1, 1, 0, 0, 1)*, *end: datetime = datetime.datetime(2022, 6, 20, 12, 16, 24, 942910)*) → List[*MergeRequest*]

robota_core.remote_provider.**new_remote_provider**(*robota_config: dict*) → Optional[*RemoteProvider*]

Factory method for RemoteProvider.

robota_core.repository module

class robota_core.repository.**Branch**(*branch*, *source: str*)

Bases: object

An abstract object representing a git branch.

Variables

- **id** – Name of branch.
- **id** – Commit id that branch points to.

class robota_core.repository.**Diff**(*diff_info*, *diff_source: str*)

Bases: object

A representation of a git diff between two points in time for a single file in a git repository.

class robota_core.repository.**Event**(*event_data*)

Bases: object

A repository event.

Variables

- **date** – The date and time of the event.
- **type** – 'deleted', 'pushed to' or 'pushed new'
- **ref_type** – The thing the event concerns, 'tag', 'branch', 'commit' etc.
- **ref_name** – The name of the thing the event concerns (branch name or tag name)
- **commit_id** – A commit id associated with ref

class robota_core.repository.**GithubRepository**(*repository_source: dict*)

Bases: *Repository*

compare(*point_1: str*, *point_2: str*) → List[*Diff*]

Compare the state of the repository at two points in time. The points may be branch names, tags or commit ids.

get_events() → List[*Event*]

Return a list of Events associated with this repository.

get_file_contents(*file_path: str*, *branch: str = 'master'*) → Optional[bytes]

Get the decoded contents of a file from the repository. Works well for text files. Might explode for other file types.

list_files(*identifier: str*) → List[str]

Returns a list of file paths with file names in a repository. Identifier can be a commit or branch name. File paths are relative to the repository root.

class robota_core.repository.**GitlabRepository**(*data_source: dict*)

Bases: *Repository*

A Gitlab flavour of a repository.

Variables
project – A connection to the gitlab repository

compare(*point_1: str*, *point_2: str*) → List[*Diff*]

Compare the state of the repository at two points in time. The points may be branch names, tags or commit ids. Point 1 must be chronologically before point 2.

get_events() → List[*Event*]

Return a list of Events associated with this repository.

get_file_contents(*file_path: str*, *branch: str = 'master'*) → Optional[bytes]

Get a file directly from the repository.

list_files(*identifier: str*) → List[str]

Returns a list of file paths with file names in a repository. Identifier can be a commit or branch name. File paths are relative to the repository root.

class robota_core.repository.**LocalRepository**(*commit_source: dict*)

Bases: *Repository*

compare(*point_1: str*, *point_2: str*) → List[*Diff*]

Compare the state of the repository at two points in time. The points may be branch names, tags or commit ids.

get_events() → List[*Event*]

Return a list of Events associated with this repository.

get_file_contents(*file_path: str*, *branch: str = 'master'*) → Optional[bytes]

Get the decoded contents of a file from the repository. Works well for text files. Might explode for other file types.

list_files(*identifier: str*) → List[str]

Returns a list of file paths with file names in a repository. Identifier can be a commit or branch name. File paths are relative to the repository root.

class robota_core.repository.**Repository**(*project_url: str*)

Bases: `object`

A place where commits, tags, events, branches and files come from.

Variables

- **_branches** – A list of Branches associated with this repository.
- **_events** – A list of Events associated with this repository.
- **_diffs** – A dictionary of cached diffs associated with this repository. They are labelled in the form key = point_1 + point_2 where point_1 and point_2 are the commit SHAs or branch names that the diff describes.

abstract compare(*point_1: str*, *point_2: str*) → List[*Diff*]

Compare the state of the repository at two points in time. The points may be branch names, tags or commit ids.

get_branch(*name: str*) → Optional[*Branch*]

Get a Branch from the repository by name. If Branch does not exist, return None.

get_branches() → List[*Branch*]

Get all of the Branches in the repository.

get_commit_by_id(*commit_id: str*) → Optional[*Commit*]

Get a Commit by its unique ID number

get_commits(*start: Optional[datetime] = None*, *end: Optional[datetime] = None*, *branch: Optional[str] = None*) → List[*Commit*]

Get issues from the issue provider between the start date and end date.

abstract get_events() → List[*Event*]

Return a list of Events associated with this repository.

abstract get_file_contents(*file_path: str*, *branch: str = 'master'*) → Optional[bytes]

Get the decoded contents of a file from the repository. Works well for text files. Might explode for other file types.

get_tag(*name: str*, *deadline: Optional[datetime] = None*, *events: Optional[List[*Event*]] = None*) → Optional[*Tag*]

Get a git Tag by name.

Parameters

- **name** – The name of the tag to get.
- **deadline** – If provided, filters tags such that tags are only returned if they existed at deadline.
- **events** – Events corresponding to the repository, required if deadline is specified.

Returns

The Tag if found else returns None.

get_tags()

Get all tags from the server.

abstract list_files(*identifier: str*) → List[str]

Returns a list of file paths with file names in a repository. Identifier can be a commit or branch name. File paths are relative to the repository root.

robota_core.repository.**new_repository**(*robota_config: dict*) → Optional[*Repository*]

Factory method for Repositories.

robota_core.string_processing module

robota_core.string_processing.**append_list_to_dict**(*dictionary: Dict[str, list]*, *key: str*, *value: list*)

If key exists in dictionary then append value to it, else add a new key with value.

Parameters

- **dictionary** – A dictionary to add key and value to.
- **key** – Dictionary key.
- **value** – A value to append to the dictionary list.

robota_core.string_processing.**build_regex_string**(*string: str*) → str

Escape some characters and replace * and ? wildcard characters with the python regex equivalents.

robota_core.string_processing.**clean**(*text: str*) → str

Convert any HTML tags to a string representation so HTML cannot be executed.

robota_core.string_processing.**get_link**(*url: str*, *link_text: Union[str, int, float]*) → str

Create link (e.g. to a commit).

robota_core.string_processing.**html_newlines**(*text: str*) → str

Replace any newline characters in a string with html newline characters.

robota_core.string_processing.**list_to_html_rows**(*list_of_strings: list*) → str

Join list items with html new lines.

robota_core.string_processing.**markdownify**(*text: str*) → str

Take text in markdown format and output the formatted text with HTML markup.

robota_core.string_processing.**replace_none**(*input_list: list*, *replacement='-'*) → list

Sanitise list for display purposes.

robota_core.string_processing.**string_to_datetime**(*date: Optional[str]*, *datetime_format: str = '%Y-%m-%dT%H:%M:%S.%fZ'*) → Optional[datetime]

Convert time string (output from GitLab project attributes) to datetime.

Parameters

- **date** – A string representing the datetime.
- **datetime_format** – The format of 'date'.

Return date

The converted datetime.

```
>>> string_to_datetime('2017-12-06T08:28:32.000Z', "%Y-%m-%dT%H:%M:%S.%fZ")
datetime.datetime(2017, 12, 6, 8, 28, 32)
```

robota_core.string_processing.**sublist_to_html_rows**(*list_of_lists: list*, *empty='-'*) → list

Separate items in a sub-list by html new lines instead of commas.

Module contents

exception robota_core.**RemoteProviderError**

Bases: Exception

The error raised when there is a problem with data from a remote provider.

robota_core.**set_up_logger**()

- genindex

PYTHON MODULE INDEX

r

INDEX

S

T

U