jira-python Documentation

Release 1.0.10

Atlassian Pty Ltd.

Nov 13, 2017

Contents

Python library to work with JIRA APIs

CHAPTER 1

Installation

Contents

The easiest (and best) way to install jira-python is through pip:

```
$ pip install jira
```

This will handle the client itself as well as the requirements.

If you're going to run the client standalone, we strongly recommend using a virtualenv, which pip can also set up for you:

```
$ pip -E jira_python install jira
$ workon jira_python
```

Doing this creates a private Python "installation" that you can freely upgrade, degrade or break without putting the critical components of your system at risk.

Source packages are also available at PyPI:

> https://pypi.python.org/pypi/jira/

1.1 Dependencies

Python 2.7 and Python 3.x are both supported.

- `requests` - Kenneth Reitz's indispensable python-requests library handles the HTTP business. Usually, the latest version available at time of release is the minimum version required; at this writing, that version is 1.2.0, but any version >= 1.0.0 should work.
- `requests-oauthlib` - Used to implement OAuth. The latest version as of this writing is 0.3.3.
- `requests-kerberos` - Used to implement Kerberos.
- `ipython` - The IPython enhanced Python interpreter provides the fancy chrome used by *Issues*.

- `filemagic` - This library handles content-type autodetection for things like image uploads. This will only work on a system that provides libmagic; Mac and Unix will almost always have it preinstalled, but Windows users will have to use Cygwin or compile it natively. If your system doesn't have libmagic, you'll have to manually specify the `contentType` parameter on methods that take an image object, such as project and user avater creation.
- `pycrypto` - This is required for the RSA-SHA1 used by OAuth. Please note that it's **not** installed automatically, since it's a fairly cumbersome process in Windows. On Linux and OS X, a `pip install pycrypto` should do it.

Installing through `pip` takes care of these dependencies for you.

CHAPTER 2

Examples

Contents

Here's a quick usage example:

```
# This script shows how to use the client in anonymous mode
# against jira.atlassian.com.
from jira import JIRA
import re

# By default, the client will connect to a JIRA instance started from the␣
↪Atlassian Plugin SDK
# (see https://developer.atlassian.com/display/DOCS/
↪Installing+the+Atlassian+Plugin+SDK for details).
# Override this with the options parameter.
```

```
options = {
    'server': 'https://jira.atlassian.com'}
jira = JIRA(options)

# Get all projects viewable by anonymous users.
projects = jira.projects()

# Sort available project keys, then return the second, third, and fourth keys.
keys = sorted([project.key for project in projects])[2:5]

# Get an issue.
issue = jira.issue('JRA-1330')

# Find all comments made by Atlassians on this issue.
atl_comments = [comment for comment in issue.fields.comment.comments
                if re.search(r'@atlassian.com$', comment.author.emailAddress)]

# Add a comment to the issue.
jira.add_comment(issue, 'Comment text')

# Change the issue's summary and description.
issue.update(
    summary="I'm different!", description='Changed the summary to be different.')

# Change the issue without sending updates
issue.update(notify=False, description='Quiet summary update.')

# You can update the entire labels field like this
issue.update(labels=['AAA', 'BBB'])

# Or modify the List of existing labels. The new label is unicode with no
# spaces
issue.fields.labels.append(u'new_text')
issue.update(fields={"labels": issue.fields.labels})

# Send the issue away for good.
issue.delete()

# Linking a remote jira issue (needs applinks to be configured to work)
issue = jira.issue('JRA-1330')
issue2 = jira.issue('XX-23')  # could also be another instance
jira.add_remote_link(issue, issue2)
```

Another example shows how to authenticate with your JIRA username and password:

```
# This script shows how to connect to a JIRA instance with a
# username and password over HTTP BASIC authentication.

from collections import Counter
from jira import JIRA

# By default, the client will connect to a JIRA instance started from the␣
↪Atlassian Plugin SDK.
# See
# https://developer.atlassian.com/display/DOCS/Installing+the+Atlassian+Plugin+SDK
# for details.
jira = JIRA(basic_auth=('admin', 'admin'))     # a username/password tuple

# Get the mutable application properties for this server (requires
# jira-system-administrators permission)
props = jira.application_properties()

# Find all issues reported by the admin
```

```
issues = jira.search_issues('assignee=admin')

# Find the top three projects containing issues reported by admin
top_three = Counter(
    [issue.fields.project.key for issue in issues]).most_common(3)
```

This example shows how to work with GreenHopper:

```
# This script shows how to use the client in anonymous mode
# against jira.atlassian.com.
from jira.client import GreenHopper

# By default, the client will connect to a JIRA instance started from the␣
↪Atlassian Plugin SDK
# (see https://developer.atlassian.com/display/DOCS/
↪Installing+the+Atlassian+Plugin+SDK for details).
# Override this with the options parameter.
# GreenHopper is a plugin in a JIRA instance
options = {
    'server': 'https://jira.atlassian.com'}
gh = GreenHopper(options)

# Get all boards viewable by anonymous users.
boards = gh.boards()

# Get the sprints in a specific board
board_id = 441
print("GreenHopper board: %s (%s)" % (boards[0].name, board_id))
sprints = gh.sprints(board_id)

# List the incomplete issues in each sprint
for sprint in sprints:
    sprint_id = sprint.id
    print("Sprint: %s" % sprint.name)
    incompleted_issues = gh.incompleted_issues(board_id, sprint_id)
    print("Incomplete issues: %s" %
          ', '.join(issue.key for issue in incompleted_issues))
```

2.1 Quickstart

2.1.1 Initialization

Everything goes through the JIRA object, so make one:

```
from jira import JIRA

jira = JIRA()
```

This connects to a JIRA started on your local machine at http://localhost:2990/jira, which not coincidentally is the default address for a JIRA instance started from the Atlassian Plugin SDK.

You can manually set the JIRA server to use:

```
jac = JIRA('https://jira.atlassian.com')
```

2.1.2 Authentication

At initialization time, jira-python can optionally create an HTTP BASIC or use OAuth 1.0a access tokens for user authentication. These sessions will apply to all subsequent calls to the JIRA object.

The library is able to load the credentials from inside the ~/.netrc file, so put them there instead of keeping them in your source code.

HTTP BASIC

Pass a tuple of (username, password) to the `basic_auth` constructor argument:

```
authed_jira = JIRA(basic_auth=('username', 'password'))
```

OAuth

Pass a dict of OAuth properties to the `oauth` constructor argument:

```
# all values are samples and won't work in your code!
key_cert_data = None
with open(key_cert, 'r') as key_cert_file:
    key_cert_data = key_cert_file.read()

oauth_dict = {
    'access_token': 'd87f3hajglkjh89a97f8',
    'access_token_secret': 'a9f8ag0ehaljkhgeds90',
    'consumer_key': 'jira-oauth-consumer',
    'key_cert': key_cert_data
}
authed_jira = JIRA(oauth=oauth_dict)
```

Note: The OAuth access tokens must be obtained and authorized ahead of time through the standard OAuth dance. For interactive use, `jirashell` can perform the dance with you if you don't already have valid tokens.

Note: OAuth in Jira uses RSA-SHA1 which requires the PyCrypto library. PyCrypto is **not** installed automatically when installing jira-python. See also the *Dependencies*. section above.

- The access token and token secret uniquely identify the user.
- The consumer key must match the OAuth provider configured on the JIRA server.
- The key cert data must be the private key that matches the public key configured on the JIRA server's OAuth provider.

See https://confluence.atlassian.com/display/JIRA/Configuring+OAuth+Authentication+for+an+Application+Link for details on configuring an OAuth provider for JIRA.

Kerberos

To enable Kerberos auth, set `kerberos=True`:

```
authed_jira = JIRA(kerberos=True)
```

2.1.3 Issues

Issues are objects. You get hold of them through the JIRA object:

```
issue = jira.issue('JRA-1330')
```

Issue JSON is marshaled automatically and used to augment the returned Issue object, so you can get direct access to fields:

```
summary = issue.fields.summary          # 'Field level security permissions'
votes = issue.fields.votes.votes        # 440 (at least)
```

If you only want a few specific fields, save time by asking for them explicitly:

```
issue = jira.issue('JRA-1330', fields='summary,comment')
```

Reassign an issue:

```
# requires issue assign permission, which is different from issue editing␣
↪permission!
jira.assign_issue(issue, 'newassignee')
```

Creating issues is easy:

```
new_issue = jira.create_issue(project='PROJ_key_or_id', summary='New issue from␣
↪jira-python',
                              description='Look into this one', issuetype={'name':
↪'Bug'})
```

Or you can use a dict:

```
issue_dict = {
    'project': {'id': 123},
    'summary': 'New issue from jira-python',
    'description': 'Look into this one',
    'issuetype': {'name': 'Bug'},
}
new_issue = jira.create_issue(fields=issue_dict)
```

You can even bulk create multiple issues:

```
issue_list = [
{
    'project': {'id': 123},
    'summary': 'First issue of many',
    'description': 'Look into this one',
    'issuetype': {'name': 'Bug'},
},
{
    'project': {'key': 'FOO'},
    'summary': 'Second issue',
    'description': 'Another one',
    'issuetype': {'name': 'Bug'},
},
{
    'project': {'name': 'Bar'},
    'summary': 'Last issue',
    'description': 'Final issue of batch.',
    'issuetype': {'name': 'Bug'},
}]
issues = jira.create_issues(field_list=issue_list)
```

Note: Project, summary, description and issue type are always required when creating issues. Your JIRA may require additional fields for creating issues; see the `jira.createmeta` method for getting access to that information.

Note: Using bulk create will not throw an exception for a failed issue creation. It will return a list of dicts that each contain a possible error signature if that issue had invalid fields. Successfully created issues will contain the issue object as a value of the `issue` key.

You can also update an issue's fields with keyword arguments:

```
issue.update(summary='new summary', description='A new summary was added')
issue.update(assignee={'name': 'new_user'})     # reassigning in update requires␣
↪issue edit permission
```

or with a dict of new field values:

```
issue.update(fields={'summary': 'new summary', 'description': 'A new summary was␣
↪added'})
```

You can suppress notifications:

```
issue.update(notify=False, description='A quiet description change was made')
```

and when you're done with an issue, you can send it to the great hard drive in the sky:

```
issue.delete()
```

Updating components:

```
existingComponents = []
for component in issue.fields.components:
    existingComponents.append({"name" : component.name})
issue.update(fields={"components": existingComponents})
```

2.1.4 Fields

issue.fields.worklogs # list of Worklog objects issue.fields.worklogs[0].author issue.fields.worklogs[0].comment issue.fields.worklogs[0].created issue.fields.worklogs[0].id issue.fields.worklogs[0].self issue.fields.worklogs[0].started issue.fields.worklogs[0].timeSpent issue.fields.worklogs[0].timeSpentSeconds issue.fields.worklogs[0].updateAuthor # dictionary issue.fields.worklogs[0].updated

issue.fields.timetracking.remainingEstimate # may be NULL or string ("0m", "2h"...) issue.fields.timetracking.remainingEstimateSeconds # may be NULL or integer issue.fields.timetracking.timeSpent # may be NULL or string issue.fields.timetracking.timeSpentSeconds # may be NULL or integer

2.1.5 Searching

Leverage the power of JQL to quickly find the issues you want:

```
issues_in_proj = jira.search_issues('project=PROJ')
all_proj_issues_but_mine = jira.search_issues('project=PROJ and assignee !=␣
↪currentUser()')

# my top 5 issues due by the end of the week, ordered by priority
```

```
oh_crap = jira.search_issues('assignee = currentUser() and due < endOfWeek() order␣
↪by priority desc', maxResults=5)

# Summaries of my last 3 reported issues
print [issue.fields.summary for issue in jira.search_issues('reporter =␣
↪currentUser() order by created desc', maxResults=3)]
```

2.1.6 Comments

Comments, like issues, are objects. Get at issue comments through the parent Issue object or the JIRA object's dedicated method:

```
comments_a = issue.fields.comment.comments
comments_b = jira.comments(issue) # comments_b == comments_a
```

Get an individual comment if you know its ID:

```
comment = jira.comment('JRA-1330', '10234')
```

Adding, editing and deleting comments is similarly straightforward:

```
comment = jira.add_comment('JRA-1330', 'new comment')    # no Issue object required
comment = jira.add_comment(issue, 'new comment', visibility={'type': 'role', 'value
↪': 'Administrators'})  # for admins only

comment.update(body = 'updated comment body')
comment.delete()
```

2.1.7 Transitions

Learn what transitions are available on an issue:

```
issue = jira.issue('PROJ-1')
transitions = jira.transitions(issue)
[(t['id'], t['name']) for t in transitions]    # [(u'5', u'Resolve Issue'), (u'2',␣
↪u'Close Issue')]
```

Note: Only the transitions available to the currently authenticated user will be returned!

Then perform a transition on an issue:

```
# Resolve the issue and assign it to 'pm_user' in one step
jira.transition_issue(issue, '5', assignee={'name': 'pm_user'}, resolution={'id':
↪'3'})

# The above line is equivalent to:
jira.transition_issue(issue, '5', fields: {'assignee':{'name': 'pm_user'},
↪'resolution':{'id': '3'}})
```

2.1.8 Projects

Projects are objects, just like issues:

```
projects = jira.projects()
```

Also, just like issue objects, project objects are augmented with their fields:

```
jra = jira.project('JRA')
print(jra.name)                 # 'JIRA'
print(jra.lead.displayName)     # 'Paul Slade [Atlassian]'
```

It's no trouble to get the components, versions or roles either (assuming you have permission):

```
components = jira.project_components(jra)
[c.name for c in components]                # 'Accessibility', 'Activity Stream',
↪'Administration', etc.

jira.project_roles(jra)                     # 'Administrators', 'Developers', etc.

versions = jira.project_versions(jra)
[v.name for v in reversed(versions)]        # '5.1.1', '5.1', '5.0.7', '5.0.6',␣
↪etc.
```

2.1.9 Watchers

Watchers are objects, represented by `jira.resources.Watchers`:

```
watcher = jira.watchers(issue)
print("Issue has {} watcher(s)".format(watcher.watchCount))
for watcher in watcher.watchers:
    print(watcher)
    # watcher is instance of jira.resources.User:
    print(watcher.emailAddress)
```

You can add users to watchers by their name:

```
jira.add_watcher(issue, 'username')
jira.add_watcher(issue, user_resource.name)
```

And of course you can remove users from watcher:

```
jira.remove_watcher(issue, 'username')
jira.remove_watcher(issue, user_resource.name)
```

2.1.10 Attachments

Attachments let user add files to issues. First you'll need an issue to which the attachment will be uploaded. Next, you'll need file itself, that is going to be attachment. File could be file-like object or string, representing path on local machine. Also you can select final name of the attachment if you don't like original. Here are some examples:

```
# upload file from `/some/path/attachment.txt`
jira.add_attachment(issue=issue, attachment='/some/path/attachment.txt')

# read and upload a file (note binary mode for opening, it's important):
with open('/some/path/attachment.txt', 'rb') as f:
    jira.add_attachment(issue=issue, attachment=f)

# attach file from memory (you can skip IO operations). In this case you MUST␣
↪provide `filename`.
import StringIO
attachment = StringIO.StringIO()
attachment.write(data)
jira.add_attachment(issue=issue, attachment=attachment, filename='content.txt')
```

If you would like to list all available attachment, you can do it with through attachment field:

```
for attachment in issue.fields.attachment:
    print("Name: '{filename}', size: {size}".format(
        filename=attachment.filename, size=attachment.size))
    # to read content use `get` method:
    print("Content: '{}'".format(attachment.get()))
```

You can delete attachment by id:

```
# Find issues with attachments:
query = jira.search_issues(jql_str="attachments is not EMPTY", json_result=True,␣
↪fields="key, attachment")

# And remove attachments one by one
for i in query['issues']:
    for a in i['fields']['attachment']:
        print("For issue {0}, found attach: '{1}' [{2}].".format(i['key'], a[
↪'filename'], a['id']))
        jira.delete_attachment(a['id'])
```

CHAPTER 3

jirashell

There is no substitute for play. The only way to really know a service, an API or a package is to explore it, poke at it, and bang your elbows – trial and error. A REST design is especially well-suited to active exploration, and the `jirashell` script (installed automatically when you use pip) is designed to help you do exactly that.

Run it from the command line:

```
$ jirashell -s http://jira.atlassian.com
<JIRA Shell (http://jira.atlassian.com)>

*** JIRA shell active; client is in 'jira'. Press Ctrl-D to exit.

In [1]:
```

This is a specialized Python interpreter (built on IPython) that lets you explore JIRA as a service. Any legal Python code is acceptable input. The shell builds a JIRA client object for you (based on the launch parameters) and stores it in the `jira` object.

Try getting an issue:

```
In [1]: issue = jira.issue('JRA-1330')
```

`issue` now contains a reference to an issue `Resource`. To see the available properties and methods, hit the TAB key:

```
In [2]: issue.
issue.delete  issue.fields  issue.id      issue.raw     issue.update
issue.expand  issue.find    issue.key     issue.self

In [2]: issue.fields.
issue.fields.aggregateprogress              issue.fields.customfield_11531
issue.fields.aggregatetimeestimate          issue.fields.customfield_11631
issue.fields.aggregatetimeoriginalestimate  issue.fields.customfield_11930
issue.fields.aggregatetimespent             issue.fields.customfield_12130
issue.fields.assignee                       issue.fields.customfield_12131
issue.fields.attachment                     issue.fields.description
issue.fields.comment                        issue.fields.environment
issue.fields.components                     issue.fields.fixVersions
issue.fields.created                        issue.fields.issuelinks
issue.fields.customfield_10150              issue.fields.issuetype
issue.fields.customfield_10160              issue.fields.labels
```

```
issue.fields.customfield_10161              issue.fields.mro
issue.fields.customfield_10180              issue.fields.progress
issue.fields.customfield_10230              issue.fields.project
issue.fields.customfield_10575              issue.fields.reporter
issue.fields.customfield_10610              issue.fields.resolution
issue.fields.customfield_10650              issue.fields.resolutiondate
issue.fields.customfield_10651              issue.fields.status
issue.fields.customfield_10680              issue.fields.subtasks
issue.fields.customfield_10723              issue.fields.summary
issue.fields.customfield_11130              issue.fields.timeestimate
issue.fields.customfield_11230              issue.fields.timeoriginalestimate
issue.fields.customfield_11431              issue.fields.timespent
issue.fields.customfield_11433              issue.fields.updated
issue.fields.customfield_11434              issue.fields.versions
issue.fields.customfield_11435              issue.fields.votes
issue.fields.customfield_11436              issue.fields.watches
issue.fields.customfield_11437              issue.fields.workratio
```

Since the *Resource* class maps the server's JSON response directly into a Python object with attribute access, you can see exactly what's in your resources.

CHAPTER 4

Advanced

4.1 Resource Objects and Properties

The library distinguishes between two kinds of data in the JIRA REST API: *resources* and *properties*.

A *resource* is a REST entity that represents the current state of something that the server owns; for example, the issue called "ABC-123" is a concept managed by JIRA which can be viewed as a resource obtainable at the URL *http://jira-server/rest/api/latest/issue/ABC-123*. All resources have a *self link*: a root-level property called *self* which contains the URL the resource originated from. In jira-python, resources are instances of the *Resource* object (or one of its subclasses) and can only be obtained from the server using the `find()` method. Resources may be connected to other resources: the issue *Resource* is connected to a user *Resource* through the `assignee` and `reporter` fields, while the project *Resource* is connected to a project lead through another user *Resource*.

Important: A resource is connected to other resources, and the client preserves this connection. In the above example, the object inside the `issue` object at `issue.fields.assignee` is not just a dict – it is a full-fledged user *Resource* object. Whenever a resource contains other resources, the client will attempt to convert them to the proper subclass of *Resource*.

A *properties object* is a collection of values returned by JIRA in response to some query from the REST API. Their structure is freeform and modeled as a Python dict. Client methods return this structure for calls that do not produce resources. For example, the properties returned from the URL *http://jira-server/rest/api/latest/issue/createmeta* are designed to inform users what fields (and what values for those fields) are required to successfully create issues in the server's projects. Since these properties are determined by JIRA's configuration, they are not resources.

The JIRA client's methods document whether they will return a *Resource* or a properties object.

CHAPTER 5

Contributing

The client is an open source project under the BSD license. Contributions of any kind are welcome!

https://github.com/pycontribs/jira/

If you find a bug or have an idea for a useful feature, file it at that bitbucket project. Extra points for source code patches – fork and send a pull request.

5.1 Discussion and support

We encourage all who wish to discuss by using https://answers.atlassian.com/questions/topics/754366/jira-python

Keep in mind to use the jira-python tag when you add a new question. This will assure that the project mantainers will get notified about your question.

CHAPTER 6

API Documentation

Contents

6.1 JIRA

class jira.**JIRA**(*server=None*, *options=None*, *basic_auth=None*, *oauth=None*, *jwt=None*, *kerberos=False*, *validate=False*, *get_server_info=True*, *async=False*, *logging=True*, *max_retries=3*, *proxies=None*, *timeout=None*)

Bases: object

User interface to JIRA.

Clients interact with JIRA by constructing an instance of this object and calling its methods. For addressable resources in JIRA – those with "self" links – an appropriate subclass of `Resource` will be returned with customized `update()` and `delete()` methods, along with attribute access to fields. This means that calls of the form `issue.fields.summary` will be resolved into the proper lookups to return the JSON value at that mapping. Methods that do not return resources will return a dict constructed from the JSON response or a scalar value; see each method's documentation for details on what that method returns.

AGILE_BASE_URL = '{server}/rest/{agile_rest_path}/{agile_rest_api_version}/{path}'

DEFAULT_OPTIONS = {'context_path': '/', 'client_cert': None, 'server': 'http://localhost:2990/jira', 'headers': {'Cac

JIRA_BASE_URL = '{server}/rest/{rest_path}/{rest_api_version}/{path}'

add_attachment(*issue*, *attachment*, *filename=None*)

Attach an attachment to an issue and returns a Resource for it.

The client will *not* attempt to open or validate the attachment; it expects a file-like object to be ready for its use. The user is still responsible for tidying up (e.g., closing the file, killing the socket, etc.)

Parameters

- **issue** – the issue to attach the attachment to
- **attachment** – file-like object to attach to the issue, also works if it is a string with the filename.
- **filename** – optional name for the attached file. If omitted, the file object's `name` attribute is used. If you aquired the file-like object by any other method than `open()`, make sure that a name is specified in one way or the other.

Return type an Attachment Resource

add_comment(*issue*, *body*, *visibility=None*)
Add a comment from the current authenticated user on the specified issue and return a Resource for it.

The issue identifier and comment body are required.

Parameters

- **issue** – ID or key of the issue to add the comment to
- **body** – Text of the comment to add
- **visibility** – a dict containing two entries: "type" and "value". "type" is 'role' (or 'group' if the JIRA server has configured comment visibility for groups) and 'value' is the name of the role (or group) to which viewing of this comment will be restricted.

add_group(*groupname*)
Create a new group in JIRA.

Parameters **groupname** – The name of the group you wish to create.

Returns Boolean - True if succesfull.

add_issues_to_epic(*epic_id*, *issue_keys*, *ignore_epics=True*)
Add the issues in `issue_keys` to the `epic_id`.

Parameters

- **epic_id** – the epic to add issues to
- **issue_keys** – the issues to add to the epic
- **ignore_epics** – ignore any issues listed in `issue_keys` that are epics

add_issues_to_sprint(*sprint_id*, *issue_keys*)
Add the issues in `issue_keys` to the `sprint_id`.

The sprint must be started but not completed.

If a sprint was completed, then have to also edit the history of the issue so that it was added to the sprint before it was completed, preferably before it started. A completed sprint's issues also all have a resolution set before the completion date.

If a sprint was not started, then have to edit the marker and copy the rank of each issue too.

Parameters

- **sprint_id** – the sprint to add issues to
- **issue_keys** – the issues to add to the sprint

add_remote_link(*issue*, *destination*, *globalId=None*, *application=None*, *relationship=None*)
Add a remote link from an issue to an external application and returns a remote link Resource for it.

`object` should be a dict containing at least `url` to the linked external URL and `title` to display for the link inside JIRA.

For definitions of the allowable fields for `object` and the keyword arguments `globalId`, `application` and `relationship`, see https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+for+Remote+Issue+Links.

Parameters

- **issue** – the issue to add the remote link to
- **destination** – the link details to add (see the above link for details)
- **globalId** – unique ID for the link (see the above link for details)
- **application** – application information for the link (see the above link for details)
- **relationship** – relationship description for the link (see the above link for details)

add_simple_link(*issue*, *object*)
Add a simple remote link from an issue to web resource.

This avoids the admin access problems from add_remote_link by just using a simple object and presuming all fields are correct and not requiring more complex `application` data.

object should be a dict containing at least url to the linked external URL and `title` to display for the link inside JIRA.

For definitions of the allowable fields for `object` , see https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+for+Remote+Issue+Links.

Parameters

- **issue** – the issue to add the remote link to
- **object** – the dictionary used to create remotelink data

add_user(*username*, *email*, *directoryId=1*, *password=None*, *fullname=None*, *notify=False*, *active=True*, *ignore_existing=False*)
Create a new JIRA user.

Parameters

- **username** (`str`) – the username of the new user
- **email** (`str`) – email address of the new user
- **directoryId** (`int`) – the directory ID the new user should be a part of
- **password** (`str`) – Optional, the password for the new user
- **fullname** (`str`) – Optional, the full name of the new user
- **notify** (`bool`) – Whether or not to send a notification to the new user
- **active** (`bool`) – Whether or not to make the new user active upon creation

add_user_to_group(*username*, *group*)
Add a user to an existing group.

Parameters

- **username** – Username that will be added to specified group.
- **group** – Group that the user will be added to.

Returns json response from Jira server for success or a value that evaluates as False in case of failure.

add_vote(*issue*)
Register a vote for the current authenticated user on an issue.

Parameters **issue** – ID or key of the issue to vote on

add_watcher(*issue*, *watcher*)
Add a user to an issue's watchers list.

Parameters

- **issue** – ID or key of the issue affected
- **watcher** – username of the user to add to the watchers list

add_worklog(*issue*, *timeSpent=None*, *timeSpentSeconds=None*, *adjustEstimate=None*, *newEstimate=None*, *reduceBy=None*, *comment=None*, *started=None*, *user=None*)
Add a new worklog entry on an issue and return a Resource for it.

Parameters

- **issue** – the issue to add the worklog to
- **timeSpent** – a worklog entry with this amount of time spent, e.g. "2d"
- **adjustEstimate** – (optional) allows the user to provide specific instructions to update the remaining time estimate of the issue. The value can either be `new`, `leave`, `manual` or `auto` (default).
- **newEstimate** – the new value for the remaining estimate field. e.g. "2d"
- **reduceBy** – the amount to reduce the remaining estimate by e.g. "2d"
- **started** – Moment when the work is logged, if not specified will default to now
- **comment** – optional worklog comment

application_properties(*key=None*)
Return the mutable server application properties.

Parameters **key** – the single property to return a value for

applicationlinks(*cached=True*)
List of application links.

Returns json

assign_issue(*issue*, *assignee*)
Assign an issue to a user. None will set it to unassigned. -1 will set it to Automatic.

Parameters

- **issue** – the issue to assign
- **assignee** – the user to assign the issue to

async_do(*size=10*)
Execute all async jobs and wait for them to finish. By default it will run on 10 threads.

Parameters **size** – number of threads to run on.

attachment(*id*)
Get an attachment Resource from the server for the specified ID.

attachment_meta()
Get the attachment metadata.

backup(*filename='backup.zip'*, *attachments=False*)
Will call jira export to backup as zipped xml. Returning with success does not mean that the backup process finished.

backup_complete()
Return boolean based on 'alternativePercentage' and 'size' returned from backup_progress (cloud only).

backup_download(*filename=None*)
Download backup file from WebDAV (cloud only).

backup_progress()
Return status of cloud backup as a dict.

Is there a way to get progress for Server version?

boards(*startAt=0*, *maxResults=50*, *type=None*, *name=None*)
Get a list of board resources.

Parameters

- **startAt** – The starting index of the returned boards. Base index: 0.
- **maxResults** – The maximum number of boards to return per page. Default: 50
- **type** – Filters results to boards of the specified type. Valid values: scrum, kanban.
- **name** – Filters results to boards that match or partially match the specified name.

Return type ResultList[Board]

When old GreenHopper private API is used, paging is not enabled and all parameters are ignored.

checked_version = False

client_info()
Get the server this client is connected to.

comment(*issue*, *comment*)
Get a comment Resource from the server for the specified ID.

Parameters

- **issue** – ID or key of the issue to get the comment from
- **comment** – ID of the comment to get

comments(*issue*)
Get a list of comment Resources.

Parameters **issue** – the issue to get comments from

component(*id*)
Get a component Resource from the server.

Parameters **id** – ID of the component to get

component_count_related_issues(*id*)
Get the count of related issues for a component.

Parameters **id** (*integer*) – ID of the component to use

confirm_project_avatar(*project*, *cropping_properties*)
Confirm the temporary avatar image previously uploaded with the specified cropping.

After a successful registry with *create_temp_project_avatar()*, use this method to confirm the avatar for use. The final avatar can be a subarea of the uploaded image, which is customized with the `cropping_properties`: the return value of *create_temp_project_avatar()* should be used for this argument.

Parameters

- **project** – ID or key of the project to confirm the avatar in
- **cropping_properties** – a dict of cropping properties from *create_temp_project_avatar()*

confirm_user_avatar(*user*, *cropping_properties*)
Confirm the temporary avatar image previously uploaded with the specified cropping.

After a successful registry with *create_temp_user_avatar()*, use this method to confirm the avatar for use. The final avatar can be a subarea of the uploaded image, which is customized with the `cropping_properties`: the return value of *create_temp_user_avatar()* should be used for this argument.

Parameters

- **user** – the user to confirm the avatar for
- **cropping_properties** – a dict of cropping properties from *create_temp_user_avatar()*

create_board(*name*, *project_ids*, *preset='scrum'*)
Create a new board for the `project_ids`.

Parameters

- **name** – name of the board
- **project_ids** – the projects to create the board in
- **preset** (*'kanban', 'scrum', 'diy'*) – what preset to use for this board

create_component(*name*, *project*, *description=None*, *leadUserName=None*, *assigneeType=None*, *isAssigneeTypeValid=False*)
Create a component inside a project and return a Resource for it.

Parameters

- **name** – name of the component
- **project** – key of the project to create the component in
- **description** – a description of the component
- **leadUserName** – the username of the user responsible for this component
- **assigneeType** – see the ComponentBean.AssigneeType class for valid values
- **isAssigneeTypeValid** – boolean specifying whether the assignee type is acceptable

create_filter(*name=None*, *description=None*, *jql=None*, *favourite=None*)
Create a new filter and return a filter Resource for it.

Parameters

- **name** – name of the new filter
- **description** – useful human readable description of the new filter
- **jql** – query string that defines the filter
- **favourite** – whether to add this filter to the current user's favorites

create_issue(*fields=None*, *prefetch=True*, ***fieldargs*)
Create a new issue and return an issue Resource for it.

Each keyword argument (other than the predefined ones) is treated as a field name and the argument's value is treated as the intended value for that field – if the fields argument is used, all other keyword arguments will be ignored.

By default, the client will immediately reload the issue Resource created by this method in order to return a complete Issue object to the caller; this behavior can be controlled through the 'prefetch' argument.

JIRA projects may contain many different issue types. Some issue screens have different requirements for fields in a new issue. This information is available through the 'createmeta' method. Further examples are available here: https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+Example+-+Create+Issue

Parameters

- **fields** – a dict containing field names and the values to use. If present, all other keyword arguments will be ignored
- **prefetch** – whether to reload the created issue Resource so that all of its data is present in the value returned from this method

create_issue_link(*type*, *inwardIssue*, *outwardIssue*, *comment=None*)
Create a link between two issues.

Parameters

- **type** – the type of link to create
- **inwardIssue** – the issue to link from
- **outwardIssue** – the issue to link to
- **comment** – a comment to add to the issues with the link. Should be a dict containing `body` and `visibility` fields: `body` being the text of the comment and `visibility` being a dict containing two entries: `type` and `value`. `type` is `role` (or `group` if the JIRA server has configured comment visibility for groups) and `value` is the name of the role (or group) to which viewing of this comment will be restricted.

create_issues(*field_list*, *prefetch=True*)
Bulk create new issues and return an issue Resource for each successfully created issue.

See *create_issue* documentation for field information.

Parameters

- **field_list** – a list of dicts each containing field names and the values to use. Each dict is an individual issue to create and is subject to its minimum requirements.
- **prefetch** – whether to reload the created issue Resource for each created issue so that all of its data is present in the value returned from this method.

create_project(*key*, *name=None*, *assignee=None*, *type='Software'*, *template_name=None*)
Key is mandatory and has to match JIRA project key requirements, usually only 2-10 uppercase characters.

If name is not specified it will use the key value. If assignee is not specified it will use current user. Parameter template_name is used to create a project based on one of the existing project templates. If template_name is not specified, then it should use one of the default values. The returned value should evaluate to False if it fails otherwise it will be the new project id.

create_sprint(*name*, *board_id*, *startDate=None*, *endDate=None*)
Create a new sprint for the `board_id`.

Parameters

- **name** – name of the sprint
- **board_id** – the board to add the sprint to

create_temp_project_avatar(*project*, *filename*, *size*, *avatar_img*, *contentType=None*, *auto_confirm=False*)
Register an image file as a project avatar.

The avatar created is temporary and must be confirmed before it can be used.

Avatar images are specified by a filename, size, and file object. By default, the client will attempt to autodetect the picture's content type: this mechanism relies on libmagic and will not work out of the box on Windows systems (see http://filemagic.readthedocs.org/en/latest/guide.html for details on how to install support). The `contentType` argument can be used to explicitly set the value (note that JIRA will reject any type other than the well-known ones for images, e.g. `image/jpg`, `image/png`, etc.)

This method returns a dict of properties that can be used to crop a subarea of a larger image for use. This dict should be saved and passed to *confirm_project_avatar()* to finish the avatar creation process. If you want to cut out the middleman and confirm the avatar with JIRA's default cropping, pass the 'auto_confirm' argument with a truthy value and *confirm_project_avatar()* will be called for you before this method returns.

Parameters

- **project** – ID or key of the project to create the avatar in
- **filename** – name of the avatar file
- **size** – size of the avatar file
- **avatar_img** – file-like object holding the avatar
- **contentType** – explicit specification for the avatar image's content-type
- **auto_confirm** (*boolean*) – whether to automatically confirm the temporary avatar by calling *confirm_project_avatar()* with the return value of this method.

create_temp_user_avatar(*user*, *filename*, *size*, *avatar_img*, *contentType=None*, *auto_confirm=False*)

Register an image file as a user avatar.

The avatar created is temporary and must be confirmed before it can be used.

Avatar images are specified by a filename, size, and file object. By default, the client will attempt to autodetect the picture's content type: this mechanism relies on `libmagic` and will not work out of the box on Windows systems (see http://filemagic.readthedocs.org/en/latest/guide.html for details on how to install support). The `contentType` argument can be used to explicitly set the value (note that JIRA will reject any type other than the well-known ones for images, e.g. `image/jpg`, `image/png`, etc.)

This method returns a dict of properties that can be used to crop a subarea of a larger image for use. This dict should be saved and passed to *confirm_user_avatar()* to finish the avatar creation process. If you want to cut out the middleman and confirm the avatar with JIRA's default cropping, pass the `auto_confirm` argument with a truthy value and *confirm_user_avatar()* will be called for you before this method returns.

Parameters

- **user** – user to register the avatar for
- **filename** – name of the avatar file
- **size** – size of the avatar file
- **avatar_img** – file-like object containing the avatar
- **contentType** – explicit specification for the avatar image's content-type
- **auto_confirm** – whether to automatically confirm the temporary avatar by calling *confirm_user_avatar()* with the return value of this method.

create_version(*name*, *project*, *description=None*, *releaseDate=None*, *startDate=None*, *archived=False*, *released=False*)

Create a version in a project and return a Resource for it.

Parameters

- **name** – name of the version to create
- **project** – key of the project to create the version in
- **description** – a description of the version
- **releaseDate** – the release date assigned to the version
- **startDate** – The start date for the version

createmeta(*projectKeys=None*, *projectIds=[]*, *issuetypeIds=None*, *issuetypeNames=None*, *expand=None*)

Get the metadata required to create issues, optionally filtered by projects and issue types.

Parameters

- **projectKeys** – keys of the projects to filter the results with. Can be a single value or a comma-delimited string. May be combined with projectIds.
- **projectIds** – IDs of the projects to filter the results with. Can be a single value or a comma-delimited string. May be combined with projectKeys.
- **issuetypeIds** – IDs of the issue types to filter the results with. Can be a single value or a comma-delimited string. May be combined with issuetypeNames.
- **issuetypeNames** – Names of the issue types to filter the results with. Can be a single value or a comma-delimited string. May be combined with issuetypeIds.
- **expand** – extra information to fetch inside each resource.

current_user()

custom_field_option(*id*)
Get a custom field option Resource from the server.

Parameters **id** – ID of the custom field to use

dashboard(*id*)
Get a dashboard Resource from the server.

Parameters **id** – ID of the dashboard to get.

dashboards(*filter=None*, *startAt=0*, *maxResults=20*)
Return a ResultList of Dashboard resources and a `total` count.

Parameters

- **filter** – either "favourite" or "my", the type of dashboards to return
- **startAt** – index of the first dashboard to return
- **maxResults** – maximum number of dashboards to return. If maxResults evaluates as False, it will try to get all items in batches.

Return type ResultList

deactivate_user(*username*)
Disable/deactivate the user.

delete_attachment(*id*)
Delete attachment by id.

Parameters **id** – ID of the attachment to delete

delete_board(*id*)
Delete an agile board.

delete_issue_link(*id*)
Delete a link between two issues.

Parameters **id** – ID of the issue link to delete

delete_project(*pid*)
Delete project from Jira.

Parameters **pid** (*str*) – JIRA projectID or Project or slug

Returns bool True if project was deleted

Raises

- ***JIRAError*** – If project not found or not enough permissions
- **ValueError** – If pid parameter is not Project, slug or ProjectID

delete_project_avatar(*project*, *avatar*)
Delete a project's avatar.

Parameters

- **project** – ID or key of the project to delete the avatar from
- **avatar** – ID of the avater to delete

delete_user(*username*)

delete_user_avatar(*username*, *avatar*)
Delete a user's avatar.

Parameters

- **username** – the user to delete the avatar from
- **avatar** – ID of the avatar to remove

editmeta(*issue*)
Get the edit metadata for an issue.

Parameters **issue** – the issue to get metadata for

email_user(*user*, *body*, *title='JIRA Notification'*)
(Obsolete) Send an email to an user via CannedScriptRunner.

favourite_filters()
Get a list of filter Resources which are the favourites of the currently authenticated user.

fields()
Return a list of all issue fields.

filter(*id*)
Get a filter Resource from the server.

Parameters **id** – ID of the filter to get.

find(*resource_format*, *ids=None*)
Find Resource object for any addressable resource on the server.

This method is a universal resource locator for any RESTful resource in JIRA. The argument `resource_format` is a string of the form `resource`, `resource/{0}`, `resource/{0}/sub`, `resource/{0}/sub/{1}`, etc. The format placeholders will be populated from the `ids` argument if present. The existing authentication session will be used.

The return value is an untyped Resource object, which will not support specialized `Resource.update()` or `Resource.delete()` behavior. Moreover, it will not know to return an issue Resource if the client uses the resource issue path. For this reason, it is intended to support resources that are not included in the standard Atlassian REST API.

Parameters

- **resource_format** – the subpath to the resource string
- **ids** (*tuple or None*) – values to substitute in the `resource_format` string

find_transitionid_by_name(*issue*, *transition_name*)
Get a transitionid available on the specified issue to the current user.

Look at https://developer.atlassian.com/static/rest/jira/6.1.html#d2e1074 for json reference

Parameters

- **issue** – ID or key of the issue to get the transitions from
- **trans_name** – iname of transition we are looking for

get_igrid(*issueid*, *customfield*, *schemeid*)

group_members(*group*)
Return a hash or users with their information. Requires JIRA 6.0 or will raise NotImplemented.

groups(*query=None*, *exclude=None*, *maxResults=9999*)
Return a list of groups matching the specified criteria.

Parameters

- **query** – filter groups by name with this string
- **exclude** – filter out groups by name with this string
- **maxResults** – maximum results to return. defaults to 9999

incompletedIssuesEstimateSum(*board_id*, *sprint_id*)
Return the total incompleted points this sprint.

issue(*id*, *fields=None*, *expand=None*)
Get an issue Resource from the server.

Parameters

- **id** – ID or key of the issue to get
- **fields** – comma-separated string of issue fields to include in the results
- **expand** – extra information to fetch inside each resource

issue_link(*id*)
Get an issue link Resource from the server.

Parameters **id** – ID of the issue link to get

issue_link_type(*id*)
Get an issue link type Resource from the server.

Parameters **id** – ID of the issue link type to get

issue_link_types()
Get a list of issue link type Resources from the server.

issue_type(*id*)
Get an issue type Resource from the server.

Parameters **id** – ID of the issue type to get

issue_type_by_name(*name*)

issue_types()
Get a list of issue type Resources from the server.

kill_session()
Destroy the session of the current authenticated user.

kill_websudo()
Destroy the user's current WebSudo session.

Works only for non-cloud deployments, for others does nothing.

move_to_backlog(*issue_keys*)
Move issues in `issue_keys` to the backlog, removing them from all sprints that have not been completed.

Parameters **issue_keys** – the issues to move to the backlog

move_version(*id*, *after=None*, *position=None*)
Move a version within a project's ordered version list and return a new version Resource for it.

One, but not both, of `after` and `position` must be specified.

Parameters

- **id** – ID of the version to move
- **after** – the self attribute of a version to place the specified version after (that is, higher in the list)
- **position** – the absolute position to move this version to: must be one of `First`, `Last`, `Earlier`, or `Later`

my_permissions(*projectKey=None*, *projectId=None*, *issueKey=None*, *issueId=None*)
Get a dict of all available permissions on the server.

Parameters

- **projectKey** – limit returned permissions to the specified project
- **projectId** – limit returned permissions to the specified project
- **issueKey** – limit returned permissions to the specified issue
- **issueId** – limit returned permissions to the specified issue

myself()
Get a dict of server information for this JIRA instance.

priorities()
Get a list of priority Resources from the server.

priority(*id*)
Get a priority Resource from the server.

Parameters **id** – ID of the priority to get

project(*id*)
Get a project Resource from the server.

Parameters **id** – ID or key of the project to get

project_avatars(*project*)
Get a dict of all avatars for a project visible to the current authenticated user.

Parameters **project** – ID or key of the project to get avatars for

project_components(*project*)
Get a list of component Resources present on a project.

Parameters **project** – ID or key of the project to get components from

project_role(*project*, *id*)
Get a role Resource.

Parameters

- **project** – ID or key of the project to get the role from
- **id** – ID of the role to get

project_roles(*project*)
Get a dict of role names to resource locations for a project.

Parameters **project** – ID or key of the project to get roles from

project_versions(*project*)
Get a list of version Resources present on a project.

Parameters **project** – ID or key of the project to get versions from

projects()
Get a list of project Resources from the server visible to the current authenticated user.

rank(*issue*, *next_issue*)
Rank an issue before another using the default Ranking field, the one named 'Rank'.

Parameters

- **issue** – issue key of the issue to be ranked before the second one.
- **next_issue** – issue key of the second issue.

reindex(*force=False*, *background=True*)
Start jira re-indexing. Returns True if reindexing is in progress or not needed, or False.

If you call reindex() without any parameters it will perform a backfround reindex only if JIRA thinks it should do it.

Parameters

- **force** – reindex even if JIRA doesn'tt say this is needed, False by default.
- **background** – reindex inde background, slower but does not impact the users, defaults to True.

remote_link(*issue*, *id*)
Get a remote link Resource from the server.

Parameters

- **issue** – the issue holding the remote link
- **id** – ID of the remote link

remote_links(*issue*)
Get a list of remote link Resources from an issue.

Parameters **issue** – the issue to get remote links from

remove_group(*groupname*)
Delete a group from the JIRA instance.

Parameters **groupname** – The group to be deleted from the JIRA instance.

Returns Boolean. Returns True on success.

remove_user_from_group(*username*, *groupname*)
Remove a user from a group.

Parameters

- **username** – The user to remove from the group.
- **groupname** – The group that the user will be removed from.

remove_vote(*issue*)
Remove the current authenticated user's vote from an issue.

Parameters **issue** – ID or key of the issue to unvote on

remove_watcher(*issue*, *watcher*)
Remove a user from an issue's watch list.

Parameters

- **issue** – ID or key of the issue affected
- **watcher** – username of the user to remove from the watchers list

removedIssuesEstimateSum(*board_id*, *sprint_id*)
Return the total incompleted points this sprint.

removed_issues(*board_id*, *sprint_id*)
Return the completed issues for the sprint.

rename_user(*old_user*, *new_user*)
Rename a JIRA user. Current implementation relies on third party plugin but in the future it may use embedded JIRA functionality.

Parameters

- **old_user** – string with username login
- **new_user** – string with username login

resolution(*id*)
Get a resolution Resource from the server.

Parameters **id** – ID of the resolution to get

resolutions()
Get a list of resolution Resources from the server.

search_allowed_users_for_issue(*user*, *issueKey=None*, *projectKey=None*, *startAt=0*, *maxResults=50*)
Get a list of user Resources that match a username string and have browse permission for the issue or project.

Parameters

- **user** – a string to match usernames against.
- **issueKey** – find users with browse permission for this issue.
- **projectKey** – find users with browse permission for this project.
- **startAt** – index of the first user to return.
- **maxResults** – maximum number of users to return. If maxResults evaluates as False, it will try to get all items in batches.

search_assignable_users_for_issues(*username*, *project=None*, *issueKey=None*, *expand=None*, *startAt=0*, *maxResults=50*)
Get a list of user Resources that match the search string for assigning or creating issues.

This method is intended to find users that are eligible to create issues in a project or be assigned to an existing issue. When searching for eligible creators, specify a project. When searching for eligible assignees, specify an issue key.

Parameters

- **username** – a string to match usernames against
- **project** – filter returned users by permission in this project (expected if a result will be used to create an issue)
- **issueKey** – filter returned users by this issue (expected if a result will be used to edit this issue)
- **expand** – extra information to fetch inside each resource
- **startAt** – index of the first user to return
- **maxResults** – maximum number of users to return. If maxResults evaluates as False, it will try to get all items in batches.

search_assignable_users_for_projects(*username*, *projectKeys*, *startAt=0*, *maxResults=50*)
Get a list of user Resources that match the search string and can be assigned issues for projects.

Parameters

- **username** – a string to match usernames against
- **projectKeys** – comma-separated list of project keys to check for issue assignment permissions
- **startAt** – index of the first user to return
- **maxResults** – maximum number of users to return. If maxResults evaluates as False, it will try to get all users in batches.

search_issues(*jql_str*, *startAt=0*, *maxResults=50*, *validate_query=True*, *fields=None*, *expand=None*, *json_result=None*)
Get a ResultList of issue Resources matching a JQL search string.

Parameters

- **jql_str** – the JQL search string to use
- **startAt** – index of the first issue to return
- **maxResults** – maximum number of issues to return. Total number of results is available in the `total` attribute of the returned ResultList. If maxResults evaluates as False, it will try to get all issues in batches.
- **fields** – comma-separated string of issue fields to include in the results
- **expand** – extra information to fetch inside each resource
- **json_result** – JSON response will be returned when this parameter is set to True. Otherwise, ResultList will be returned.

search_users(*user*, *startAt=0*, *maxResults=50*, *includeActive=True*, *includeInactive=False*)
Get a list of user Resources that match the specified search string.

Parameters

- **user** – a string to match usernames, name or email against.
- **startAt** – index of the first user to return.
- **maxResults** – maximum number of users to return. If maxResults evaluates as False, it will try to get all items in batches.
- **includeActive** – If true, then active users are included in the results.
- **includeInactive** – If true, then inactive users are included in the results.

security_level(*id*)
Get a security level Resource.

Parameters **id** – ID of the security level to get

server_info()
Get a dict of server information for this JIRA instance.

session()
Get a dict of the current authenticated user's session information.

set_application_property(*key*, *value*)
Set the application property.

Parameters

- **key** – key of the property to set
- **value** – value to assign to the property

set_project_avatar(*project*, *avatar*)
Set a project's avatar.

Parameters

- **project** – ID or key of the project to set the avatar on
- **avatar** – ID of the avatar to set

set_user_avatar(*username*, *avatar*)
Set a user's avatar.

Parameters

- **username** – the user to set the avatar for
- **avatar** – ID of the avatar to set

sprint(*id*)

sprint_info(*board_id*, *sprint_id*)
Return the information about a sprint.

Parameters

- **board_id** – the board retrieving issues from. Deprecated and ignored.
- **sprint_id** – the sprint retrieving issues from

sprints(*board_id*, *extended=False*, *startAt=0*, *maxResults=50*, *state=None*)
Get a list of sprint GreenHopperResources.

Parameters

- **board_id** – the board to get sprints from
- **extended** – Used only by old GreenHopper API to fetch additional information like startDate, endDate, completeDate, much slower because it requires an additional requests for each sprint. New JIRA Agile API always returns this information without a need for additional requests.
- **startAt** – the index of the first sprint to return (0 based)
- **maxResults** – the maximum number of sprints to return
- **state** – Filters results to sprints in specified states. Valid values: future, active, closed. You can define multiple states separated by commas

Return type dict

Returns (content depends on API version, but always contains id, name, state, startDate and endDate) When old GreenHopper private API is used, paging is not enabled, and *startAt*, *maxResults* and *state* parameters are ignored.

sprints_by_name(*id*, *extended=False*)

status(*id*)
Get a status Resource from the server.

Parameters **id** – ID of the status resource to get

statuses()
Get a list of status Resources from the server.

transition_issue(*issue*, *transition*, *fields=None*, *comment=None*, ***fieldargs*)
Perform a transition on an issue.

Each keyword argument (other than the predefined ones) is treated as a field name and the argument's value is treated as the intended value for that field – if the fields argument is used, all other keyword arguments will be ignored. Field values will be set on the issue as part of the transition process.

Parameters

- **issue** – ID or key of the issue to perform the transition on
- **transition** – ID or name of the transition to perform
- **comment** – *Optional* String to add as comment to the issue when performing the transition.
- **fields** – a dict containing field names and the values to use. If present, all other keyword arguments will be ignored

transitions(*issue*, *id=None*, *expand=None*)
Get a list of the transitions available on the specified issue to the current user.

Parameters

- **issue** – ID or key of the issue to get the transitions from
- **id** – if present, get only the transition matching this ID
- **expand** – extra information to fetch inside each transition

update_filter(*filter_id*, *name=None*, *description=None*, *jql=None*, *favourite=None*)
Update a filter and return a filter Resource for it.

Parameters

- **name** – name of the new filter
- **description** – useful human readable description of the new filter
- **jql** – query string that defines the filter
- **favourite** – whether to add this filter to the current user's favorites

update_sprint(*id*, *name=None*, *startDate=None*, *endDate=None*, *state=None*)

user(*id*, *expand=None*)
Get a user Resource from the server.

Parameters

- **id** – ID of the user to get
- **expand** – extra information to fetch inside each resource

user_avatars(*username*)
Get a dict of avatars for the specified user.

Parameters **username** – the username to get avatars for

version(*id*, *expand=None*)
Get a version Resource.

Parameters

- **id** – ID of the version to get
- **expand** – extra information to fetch inside each resource

version_count_related_issues(*id*)
Get a dict of the counts of issues fixed and affected by a version.

Parameters **id** – the version to count issues for

version_count_unresolved_issues(*id*)
Get the number of unresolved issues for a version.

Parameters **id** – ID of the version to count issues for

votes(*issue*)
Get a votes Resource from the server.

Parameters **issue** – ID or key of the issue to get the votes for

watchers(*issue*)
Get a watchers Resource from the server for an issue.

Parameters **issue** – ID or key of the issue to get the watchers for

worklog(*issue*, *id*)
Get a specific worklog Resource from the server.

Parameters

- **issue** – ID or key of the issue to get the worklog from
- **id** – ID of the worklog to get

worklogs(*issue*)
Get a list of worklog Resources from the server for an issue.

Parameters **issue** – ID or key of the issue to get worklogs from

6.2 Priority

class jira.**Priority**(*options*, *session*, *raw=None*)
Bases: jira.resources.Resource

Priority that can be set on an issue.

6.3 Comment

class jira.**Comment**(*options*, *session*, *raw=None*)
Bases: jira.resources.Resource

An issue comment.

update(*fields=None*, *async=None*, *jira=None*, *body=''*, *visibility=None*)

6.4 Worklog

class jira.**Worklog**(*options*, *session*, *raw=None*)
Bases: jira.resources.Resource

Worklog on an issue.

delete(*adjustEstimate=None*, *newEstimate=None*, *increaseBy=None*)
Delete this worklog entry from its associated issue.

Parameters

- **adjustEstimate** – one of new, leave, manual or auto. auto is the default and adjusts the estimate automatically. leave leaves the estimate unchanged by this deletion.
- **newEstimate** – combined with adjustEstimate=new, set the estimate to this value
- **increaseBy** – combined with adjustEstimate=manual, increase the remaining estimate by this amount

6.5 Watchers

class jira.**Watchers**(*options*, *session*, *raw=None*)
Bases: jira.resources.Resource

Watcher information on an issue.

delete(*username*)
Remove the specified user from the watchers list.

6.6 JIRAError

class jira.**JIRAError**(*status_code=None*, *text=None*, *url=None*, *request=None*, *response=None*, ***kwargs*)
Bases: Exception

General error raised for all problems in operation of the client.

log_to_tempfile = True

This documents the `jira` python package (version 1.0.10), a Python library designed to ease the use of the JIRA REST API. Some basic support for the GreenHopper REST API also exists.

Documentation is also available in Dash format.

The source is stored at https://github.com/pycontribs/jira.

Until someone will find a better way to generate the release notes you can read https://github.com/pycontribs/jira/blob/master/CHANGELOG which is generated based on git commit messages.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index

I

J

K

L

M

P

R

S

T

U

V

W