jira-python Documentation

Release 3.8.1.dev5+geb0ec90

Atlassian Pty Ltd.

CONTENTS

1 Installation **3**
- 1.1 Dependencies . . . 3

2 Examples **5**
- 2.1 Quickstart . . . 7

3 jirashell **17**

4 Advanced **19**
- 4.1 Resource Objects and Properties . . . 19

5 Contributing **21**
- 5.1 Discussion and support . . . 21
- 5.2 Contributing Code . . . 21
- 5.3 Testing . . . 21
- 5.4 Issues and Feature Requests . . . 23
- 5.5 Issues . . . 23
- 5.6 Pull Requests . . . 23

6 API Documentation **25**
- 6.1 jira package . . . 25

7 Indices and tables **79**

Python Module Index **81**

Index **83**

Python library to work with Jira APIs

CHAPTER

ONE

INSTALLATION

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

```
pip install 'jira[cli]'
```

This will handle installation of the client itself as well as the requirements. The *[cli]* part installs dependencies for the *jirashell* binary, and may be omitted if you just need the library.

If you're going to run the client standalone, we strongly recommend using a virtualenv:

```
python -m venv jira_python
source jira_python/bin/activate
pip install 'jira[cli]'
```

or:

```
python -m venv jira_python
jira_python/bin/pip install 'jira[cli]'
```

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 >=3.8 is required.

- `requests` - 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 1.3.0.
- `requests-kerberos` - Used to implement Kerberos.
- `ipython` - The IPython enhanced Python interpreter provides the fancy chrome used by *Headers*.
- `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 avatar creation.

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

CHAPTER

TWO

EXAMPLES

Here's a quick usage example:

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

import re

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(server="https://jira.atlassian.com")

# 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.key)
]

# 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(fields={"labels": ["AAA", "BBB"]})
```

(continues on next page)

(continued from previous page)

```
# Or modify the List of existing labels. The new label is unicode with no
# spaces
issue.fields.labels.append("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.id, issue2)
```

Another example with methods to authenticate with your Jira:

```
"""Some simple authentication examples."""

from __future__ import annotations

from collections import Counter
from typing import cast

from jira import JIRA
from jira.client import ResultList
from jira.resources import Issue

# Some Authentication Methods
jira = JIRA(
    basic_auth=("admin", "admin"),  # a username/password tuple [Not recommended]
    # basic_auth=("email", "API token"),  # Jira Cloud: a username/token tuple
    # token_auth="API token",  # Self-Hosted Jira (e.g. Server): the PAT token
    # auth=("admin", "admin"),  # a username/password tuple for cookie auth [Not␣
↪recommended]
)

# Who has authenticated
myself = jira.myself()

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

# Find all issues reported by the admin
# Note: we cast() for mypy's benefit, as search_issues can also return the raw json !
#   This is if the following argument is used: `json_result=True`
issues = cast(ResultList[Issue], 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 Jira Agile / Jira Software (formerly GreenHopper):

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

from jira.client import JIRA
```

(continues on next page)

(continued from previous page)

```
# 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.
jira = JIRA(server="https://jira.atlassian.com")

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

# Get the sprints in a specific board
board_id = 441
print(f"JIRA board: {boards[0].name} ({board_id})")
sprints = jira.sprints(board_id)
```

2.1 Quickstart

2.1.1 Initialization

Everything goes through the *jira.client.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:

```
jira = 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.client.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.

Cookie Based Authentication

> **Warning:** This method of authentication is no longer supported on Jira Cloud. You can find the deprecation notice here.
>
> For Jira Cloud use the basic_auth= *(username, api_token)* authentication

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

```
auth_jira = JIRA(auth=('username', 'password'))
```

Using this method, authentication happens during the initialization of the object. If the authentication is successful, the retrieved session cookie will be used in future requests. Upon cookie expiration, authentication will happen again transparently.

HTTP BASIC

(username, password)

> **Warning:** This method of authentication is no longer supported on Jira Cloud. You can find the deprecation notice here
>
> For Jira Cloud use the basic_auth= *(username, api_token)* authentication. For Self Hosted Jira (Server, Data Center), consider the *Token Auth* authentication.

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

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

(username, api_token)

Or pass a tuple of (email, api_token) to the `basic_auth` constructor argument (JIRA Cloud):

```
auth_jira = JIRA(basic_auth=('email', 'API token'))
```

See also:

For Self Hosted Jira (Server, Data Center), refer to the *Token Auth* Section.

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': 'foo',
    'access_token_secret': 'bar',
    'consumer_key': 'jira-oauth-consumer',
    'key_cert': key_cert_data
}
auth_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.

- 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.

Token Auth

Jira Cloud

This is also referred to as an API Token in the Jira Cloud documentation

```
auth_jira = JIRA(basic_auth=('email', 'API token'))
```

Jira Self Hosted (incl. Jira Server/Data Center)

This is also referred to as Personal Access Tokens (PATs) in the Self-Hosted Documentation. The is available from Jira Core >= 8.14:

```
auth_jira = JIRA(token_auth='API token')
```

Kerberos

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

```
auth_jira = JIRA(kerberos=True)
```

To pass additional options to Kerberos auth use dict `kerberos_options`, e.g.:

```
auth_jira = JIRA(kerberos=True, kerberos_options={'mutual_authentication': 'DISABLED'}
↪)
```

2.1.3 Headers

Headers can be provided to the internally used `requests.Session`. If the user provides a header that the *jira.client.JIRA* also attempts to set, the user provided header will take preference.

For example if you want to use a custom User Agent:

```
from requests_toolbelt import user_agent

jira = JIRA(
    basic_auth=("email", "API token"),
    options={"headers": {"User-Agent": user_agent("my_package", "0.0.1")}},
)
```

2.1.4 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')
```

If you want to unassign it again, just do:

```
jira.assign_issue(issue, None)
```

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 re-

quire 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})
```

Working with Rich Text

You can use rich text in an issue's description or comment. In order to use rich text, the body content needs to be formatted using the Atlassian Document Format (ADF):

```
jira = JIRA(basic_auth=("email", "API token"))
comment = {
    "type": "doc",
    "version": 1,
    "content": [
      {
        "type": "codeBlock",
        "content": [
          {
            "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.␣
↪Pellentesque eget venenatis elit. Duis eu justo eget augue iaculis fermentum. Sed␣
↪semper quam laoreet nisi egestas at posuere augue semper.",
            "type": "text"
          }
        ]
      }
    ]
  }
jira.add_comment("AB-123", comment)
```

2.1.5 Fields

Example for accessing the worklogs:

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

2.1.6 Searching

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

```
# Search returns first 50 results, `maxResults` must be set to exceed this
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
for issue in jira.search_issues('reporter = currentUser() order by created desc',␣
↪maxResults=3):
    print('{}: {}'.format(issue.key, issue.fields.summary))
```

2.1.7 Comments

Comments, like issues, are objects. Access 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
```

Obtain an individual comment if you know its ID:

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

Obtain comment author name and comment creation timestamp if you know its ID:

```
author = jira.comment('JRA-1330', '10234').author.displayName
time = jira.comment('JRA-1330', '10234').created
```

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.update(body='updated comment body but no mail notification', notify=False)
comment.delete()
```

Get all images from a comment:

```
issue = jira.issue('JRA-1330')
regex_for_png = re.compile(r'\!(\S+?\.(jpg|png|bmp))\|?\S*?\!')
pngs_used_in_comment = regex_for_png.findall(issue.fields.comment.comments[0].body)
for attachment in issue.fields.attachment:
    if attachment.filename in pngs_used_in_comment:
        with open(attachment.filename, 'wb') as f:
            f.write(attachment.get())
```

2.1.8 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'}})
```

2.1.9 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)     # 'John Doe [ACME Inc.]'
```

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.
```

2.1.10 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)
```

2.1.11 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 the file itself that is going to be attachment. The file could be a file-like object or string, representing path on the local machine. You can also modify the 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:
```

(continues on next page)

(continued from previous page)

```
    jira.add_attachment(issue=issue, attachment=f)

# attach file from memory (you can skip IO operations). In this case you MUST provide␣
↪`filename`.
from io import StringIO
attachment = 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

THREE

JIRASHELL

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

```
pip install jira[cli]
```

Run it from the command line

```
jirashell -s https://jira.atlassian.com
<Jira Shell (https://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
```

(continues on next page)

(continued from previous page)

```
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

FOUR

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

FIVE

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 the GitHub 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://community.atlassian.com/t5/tag/jira-python/tg-p

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

5.2 Contributing Code

- **Patches should be:**
  - concise
  - work across all supported versions of Python.
  - follows the existing style of the code base (PEP-8).
  - included comments as required.
- **Great Patch has:**
  - A test case that demonstrates the previous flaw that now passes with the included patch.
  - Documentation for those changes to a public API

5.3 Testing

5.3.1 Dev Container

We utilise Docker in order to generate a test Jira Server instance.

This can be run manually, or automated using VS Code Dev Containers:

1. Open the folder of the repository with VS Code
2. Ensure you have Docker running
3. Ensure you have the `ms-azuretools.vscode-docker` and `ms-vscode-remote.remote-containers` extensions installed.

4. You should be able to do `View >> Command Palette` (or equivalent) and search for: `Remote-containers: Rebuild and Reopen in container`.

This will use the `.devcontainer\Dockerfile` as a base image with configurations dictated by `.devcontainer\devcontainer.json`.

Tip: The Docker extension can be used to monitor the progress of the Jira server build, it takes a while! The tests will only run once the server is up and reachable on: http://localhost:2990/jira

---

5.3.2 Running Tests

Using tox

```
python -m pip install pipx
pipx install tox
tox
```

- **Lint**
  - `tox -e lint`
- **Run tests**
  - `tox`
- **Run tests for one env only**
  - `tox -e py38`
- **Specify what tests to run with pytest**
  - `tox -e py39 -- tests/resources/test_attachment.py`
  - `tox -e py38 -- -m allow_on_cloud` (Run only the cloud tests)
- **Debug tests with breakpoints by disabling the coverage plugin, with the `--no-cov` argument.**
  - Example for VSCode on Windows :

```
{
    "name": "Pytest",
    "type": "python",
    "request": "launch",
    "python": ".tox\\py39\\Scripts\\python.exe",
    "module": "pytest",
    "env": {
        "CI_JIRA_URL": "http://localhost:2990/jira",
        "CI_JIRA_ADMIN": "admin",
        "CI_JIRA_ADMIN_PASSWORD": "admin",
        "CI_JIRA_USER": "jira_user",
        "CI_JIRA_USER_FULL_NAME": "Newly Created CI User",
        "CI_JIRA_USER_PASSWORD": "jira",
        "CI_JIRA_ISSUE": "Task",
        "PYTEST_TIMEOUT": "0", // Don't timeout
    },
    "args": [
        // "-v",
        "--no-cov", // running coverage affects breakpoints
        "tests/resources/test_attachment.py"
    ]
}
```

5.4 Issues and Feature Requests

- Check to see if there's an existing issue/pull request for the bug/feature. All issues are at https://github.com/pycontribs/jira/issues and pull requests are at https://github.com/pycontribs/jira/pulls.
- If there isn't an existing issue there, please file an issue.
  - An example template is provided for:
    - Bugs: https://github.com/pycontribs/jira/blob/main/.github/ISSUE_TEMPLATE/bug_report.yml
    - Features: https://github.com/pycontribs/jira/blob/main/.github/ISSUE_TEMPLATE/feature_request.yml
  - If possible, create a pull request with a (failing) test case demonstrating what's wrong. This makes the process for fixing bugs quicker & gets issues resolved sooner.

5.5 Issues

Here are the best ways to help with open issues:

- **For issues without reproduction steps**
  - Try to reproduce the issue, comment with the minimal amount of steps to reproduce the bug (a code snippet would be ideal).
  - If there is not a set of steps that can be made to reproduce the issue, at least make sure there are debug logs that capture the unexpected behavior.
- Submit pull requests for open issues.

5.6 Pull Requests

There are some key points that are needed to be met before a pull request can be merged:

- **All tests must pass for all python versions. (Once the Test Framework is fixed)**
  - For now, no new failures should occur
- All pull requests require tests that either test the new feature or test that the specific bug is fixed. Pull requests for minor things like adding a new region or fixing a typo do not need tests.
- Must follow PEP8 conventions.
- Within a major version changes must be backwards compatible.

The best way to help with pull requests is to comment on pull requests by noting if any of these key points are missing, it will both help get feedback sooner to the issuer of the pull request and make it easier to determine for an individual with write permissions to the repository if a pull request is ready to be merged.

CHAPTER

SIX

API DOCUMENTATION

6.1 jira package

6.1.1 jira.client module

Jira Client module.

This module implements a friendly (well, friendlier) interface between the raw JSON responses from Jira and the Resource/dict abstractions provided by this library. Users will construct a JIRA object as described below. Full API documentation can be found at: https://jira.readthedocs.io/en/latest/.

jira.client.**cloud_api**(*client_method: Callable*) → Callable

A convenience decorator to check if the Jira instance is cloud.

Checks if the client instance is talking to Cloud Jira. If it is, return the result of the called client method. If not, return None and log a warning.

Parameters
client_method – The method that is being called by the client.

Returns
Either the result of the wrapped function or None.

Raises

- ***JIRAError*** – In the case the error is not an HTTP error with a status code.
- ***NotJIRAInstanceError*** – In the case that the first argument to this method is not a *client.JIRA* instance.

jira.client.**experimental_atlassian_api**(*client_method: Callable*) → Callable

A convenience decorator to inform if a client method is experimental.

Indicates the path covered by the client method is experimental. If the path disappears or the method becomes disallowed, this logs an error and returns None. If another kind of exception is raised, this reraises.

Raises

- ***JIRAError*** – In the case the error is not an HTTP error with a status code.
- ***NotJIRAInstanceError*** – In the case that the first argument to this method is is not a *client.JIRA* instance.

Returns
Either the result of the wrapped function or None.

jira.client.**translate_resource_args**(*func: Callable*)

Decorator that converts Issue and Project resources to their keys when used as arguments.

Parameters
func (*Callable*) – the function to decorate

class jira.client.**ResultList**(*iterable: Iterable = None*, *_startAt: int = 0*, *_maxResults: int = 0*, *_total: int | None = None*, *_isLast: bool | None = None*)

Bases: list, Generic[*ResourceType*]

__init__(*iterable: Iterable = None*, *_startAt: int = 0*, *_maxResults: int = 0*, *_total: int | None = None*, *_isLast: bool | None = None*) → None

Results List.

Parameters

- **iterable** (*Iterable*) – [description]. Defaults to None.
- **_startAt** (*int*) – Start page. Defaults to 0.
- **_maxResults** (*int*) – Max results per page. Defaults to 0.
- **_total** (*Optional[int]*) – Total results from query. Defaults to 0.
- **_isLast** (*Optional[bool]*) – True to mark this page is the last page? (Default: None). see The official API docs

class jira.client.**QshGenerator**(*context_path*)

Bases: object

__init__(*context_path*)

class jira.client.**JiraCookieAuth**(*session: ResilientSession*, *session_api_url: str*, *auth: tuple[str, str]*)

Bases: AuthBase

Jira Cookie Authentication.

Allows using cookie authentication as described by jira api docs

__init__(*session: ResilientSession*, *session_api_url: str*, *auth: tuple[str, str]*)

Cookie Based Authentication.

Parameters

- **session** (ResilientSession) – The Session object to communicate with the API.
- **session_api_url** (*str*) – The session api url to use.
- **auth** (*Tuple[str, str]*) – The username, password tuple.

property cookies

init_session()

Initialise the Session object's cookies, so we can use the session cookie.

Raises HTTPError if the post returns an erroring http response

handle_401(*response: Response*, ***kwargs*) → Response

Refresh cookies if the session cookie has expired. Then retry the request.

Parameters
response (*requests.Response*) – the response with the possible 401 to handle

Returns
requests.Response

process_original_request(*original_request: PreparedRequest*)

update_cookies(*original_request: PreparedRequest*)

send_request(*request: PreparedRequest*)

class jira.client.**TokenAuth**(*token: str*)

Bases: AuthBase

Bearer Token Authentication.

__init__(*token: str*)

class jira.client.**JIRA**(*server: str = None*, *options: dict[str, str | bool | Any] = None*, *basic_auth: tuple[str, str] | None = None*, *token_auth: str | None = None*, *oauth: dict[str, Any] = None*, *jwt: dict[str, Any] = None*, *kerberos=False*, *kerberos_options: dict[str, Any] = None*, *validate=False*, *get_server_info: bool = True*, *async_: bool = False*, *async_workers: int = 5*, *logging: bool = True*, *max_retries: int = 3*, *proxies: Any = None*, *timeout: None | float | tuple[float, float] | tuple[float, None] = None*, *auth: tuple[str, str] = None*, *default_batch_sizes: dict[type[jira.resources.Resource], int | None] | None = 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 *jira.resources.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.

Without any arguments, this client will connect anonymously to the Jira instance started by the Atlassian Plugin SDK from one of the 'atlas-run', `atlas-debug` or `atlas-run-standalone` commands. By default, this instance runs at `http://localhost:2990/jira`. The `options` argument can be used to set the Jira instance to use.

Authentication is handled with the `basic_auth` argument. If authentication is supplied (and is accepted by Jira), the client will remember it for subsequent requests.

For quick command line access to a server, see the `jirashell` script included with this distribution.

The easiest way to instantiate is using `j = JIRA("https://jira.atlassian.com")`

DEFAULT_OPTIONS = {'agile_rest_api_version': '1.0', 'agile_rest_path': 'agile', 'async': False, 'async_workers': 5, 'auth_url': '/rest/auth/1/session', 'check_update': False, 'client_cert': None, 'context_path': '/', 'default_batch_size': {<class 'jira.resources.Resource'>: 100}, 'delay_reload': 0, 'headers': {'Cache-Control': 'no-cache', 'Content-Type': 'application/json', 'X-Atlassian-Token': 'no-check'}, 'resilient': True, 'rest_api_version': '2', 'rest_path': 'api', 'server': 'http://localhost:2990/jira', 'verify': True}

checked_version = False

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

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

__init__(*server: str = None*, *options: dict[str, str | bool | Any] = None*, *basic_auth: tuple[str, str] | None = None*, *token_auth: str | None = None*, *oauth: dict[str, Any] = None*, *jwt: dict[str, Any] = None*, *kerberos=False*, *kerberos_options: dict[str, Any] = None*, *validate=False*, *get_server_info: bool = True*, *async_: bool = False*, *async_workers: int = 5*, *logging: bool = True*, *max_retries: int = 3*, *proxies: Any = None*, *timeout: None | float | tuple[float, float] | tuple[float, None] = None*, *auth: tuple[str, str] = None*, *default_batch_sizes: dict[type[jira.resources.Resource], int | None] | None = None*)

Construct a Jira client instance.

Without any arguments, this client will connect anonymously to the Jira instance started by the Atlassian Plugin SDK from one of the 'atlas-run', `atlas-debug` or `atlas-run-standalone` commands. By default, this instance runs at `http://localhost:2990/jira`. The `options` argument can be used to set the Jira instance to use.

Authentication is handled with the `basic_auth` or `token_auth` argument. If authentication is supplied (and is accepted by Jira), the client will remember it for subsequent requests.

For quick command line access to a server, see the `jirashell` script included with this distribution.

The easiest way to instantiate is using `j = JIRA("https://jira.atlasian.com")`

Parameters

- **server** (*Optional[str]*) – The server address and context path to use. Defaults to `http://localhost:2990/jira`.
- **options** (*Optional[Dict[str, bool, Any]]*) – Specify the server and properties this client will use. Use a dict with any of the following properties:
  - server – the server address and context path to use. Defaults to `http://localhost:2990/jira`.
  - rest_path – the root REST path to use. Defaults to `api`, where the Jira REST resources live.
  - rest_api_version – the version of the REST resources under rest_path to use. Defaults to `2`.
  - agile_rest_path - the REST path to use for Jira Agile requests. Defaults to `agile`.
  - verify (Union[bool, str]) – Verify SSL certs. (Default: `True`). Or path to a CA_BUNDLE file or directory with certificates of trusted CAs, for the *requests* library to use.
  - client_cert (Union[str, Tuple[str,str]]) – Path to file with both cert and key or a tuple of (cert,key), for the *requests* library to use for client side SSL.
  - check_update – Check whether using the newest python-jira library version.
  - headers – a dict to update the default headers the session uses for all API requests.
- **basic_auth** (*Optional[Tuple[str, str]]*) – A tuple of username and password to use when establishing a session via HTTP BASIC authentication.
- **token_auth** (*Optional[str]*) – A string containing the token necessary for (PAT) bearer token authorization.
- **oauth** (*Optional[Any]*) – A dict of properties for OAuth authentication. The following properties are required:
  - access_token – OAuth access token for the user
  - access_token_secret – OAuth access token secret to sign with the key
  - consumer_key – key of the OAuth application link defined in Jira
  - key_cert – private key file to sign requests with (should be the pair of the public key supplied to Jira in the OAuth application link)
  - signature_method (Optional) – The signature method to use with OAuth. Defaults to oauthlib.oauth1.SIGNATURE_HMAC_SHA1
- **kerberos** (*bool*) – True to enable Kerberos authentication. (Default: `False`)
- **kerberos_options** (*Optional[Dict[str,str]]*) – A dict of properties for Kerberos authentication. The following properties are possible:
  - mutual_authentication – string DISABLED or OPTIONAL.

Example kerberos_options structure: `{'mutual_authentication': 'DISABLED'}`

- **jwt** (*Optional[Any]*) – A dict of properties for JWT authentication supported by Atlassian Connect. The following properties are required:
  - secret – shared secret as delivered during 'installed' lifecycle event (see https://developer.atlassian.com/static/connect/docs/latest/modules/lifecycle.html for details)
  - payload – dict of fields to be inserted in the JWT payload, e.g. 'iss'

  Example jwt structure: `{'secret': SHARED_SECRET, 'payload': {'iss': PLUGIN_KEY}}`
- **validate** (*bool*) – True makes your credentials first to be validated. Remember that if you are accessing Jira as anonymous it will fail. (Default: `False`).
- **get_server_info** (*bool*) – True fetches server version info first to determine if some API calls are available. (Default: `True`).
- **async** (*bool*) – True enables async requests for those actions where we implemented it, like issue update() or delete(). (Default: `False`).
- **async_workers** (*int*) – Set the number of worker threads for async operations.
- **timeout** (*Optional[Union[Union[float, int], Tuple[float, float]]]*) – Set a read/connect timeout for the underlying calls to Jira. Obviously this means that you cannot rely on the return code when this is enabled.
- **max_retries** (*int*) – Sets the amount Retries for the HTTP sessions initiated by the client. (Default: `3`)
- **proxies** (*Optional[Any]*) – Sets the proxies for the HTTP session.
- **auth** (*Optional[Tuple[str,str]]*) – Set a cookie auth token if this is required.
- **logging** (*bool*) – True enables loglevel to info => else critical. (Default: `True`)
- **default_batch_sizes** (*Optional[Dict[Type[Resource], Optional[int]]]*) – Manually specify the batch-sizes for the paginated retrieval of different item types. *Resource* is used as a fallback for every item type not specified. If an item type is mapped to *None* no fallback occurs, instead the JIRA-backend will use its default batch-size. By default all Resources will be queried in batches of 100. E.g., setting this to `{Issue: 500, Resource: None}` will make *search_issues()* query Issues in batches of 500, while every other item type's batch-size will be controlled by the backend. (Default: None)

property server_url: str

Return the server url.

Returns
str

close()

client_info() → str

Get the server this client is connected to.

find(*resource_format: str*, *ids: tuple[str, str] | int | str = ''*) → *Resource*

Find Resource object for any addressable resource on the server.

This method is a universal resource locator for any REST-ful 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** (*str*) – the subpath to the resource string
- **ids** (*Optional[Tuple]*) – values to substitute in the `resource_format` string

Returns
Resource

async_do(*size: int = 10*)

Execute all asynchronous jobs and wait for them to finish. By default it will run on 10 threads.

Parameters
size (*int*) – number of threads to run on.

application_properties(*key: str = None*) → dict[str, str] | list[dict[str, str]]

Return the mutable server application properties.

Parameters
key (*Optional[str]*) – the single property to return a value for

Returns
Union[Dict[str, str], List[Dict[str, str]]]

set_application_property(*key: str*, *value: str*)

Set the application property.

Parameters

- **key** (*str*) – key of the property to set
- **value** (*str*) – value to assign to the property

applicationlinks(*cached: bool = True*) → list

List of application links.

Returns
List[Dict] – json, or empty list

attachment(*id: str*) → *Attachment*

Get an attachment Resource from the server for the specified ID.

Parameters
id (*str*) – The Attachment ID

Returns
Attachment

attachment_meta() → dict[str, int]

Get the attachment metadata.

Returns
Dict[str, int]

add_attachment(*issue: str | int*, *attachment: str | BufferedReader*, *filename: str = None*) → *Attachment*

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** (*Union[str, int]*) – the issue to attach the attachment to

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

Returns
Attachment

delete_attachment(*id: str*) → Response

Delete attachment by id.

Parameters
id (*str*) – ID of the attachment to delete

Returns
Response

component(*id: str*)

Get a component Resource from the server.

Parameters
id (*str*) – ID of the component to get

create_component(*name: str*, *project: str*, *description=None*, *leadUserName=None*, *assigneeType=None*, *isAssigneeTypeValid=False*) → *Component*

Create a component inside a project and return a Resource for it.

Parameters

- **name** (*str*) – name of the component
- **project** (*str*) – key of the project to create the component in
- **description** (*str*) – a description of the component
- **leadUserName** (*Optional[str]*) – the username of the user responsible for this component
- **assigneeType** (*Optional[str]*) – see the ComponentBean.AssigneeType class for valid values
- **isAssigneeTypeValid** (*bool*) – True specifies whether the assignee type is acceptable (Default: `False`)

Returns
Component

component_count_related_issues(*id: str*)

Get the count of related issues for a component.

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

delete_component(*id: str*) → Response

Delete component by id.

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

Returns
Response

custom_field_option(*id: str*) → *CustomFieldOption*

Get a custom field option Resource from the server.

Parameters
id (*str*) – ID of the custom field to use

Returns
CustomFieldOption

dashboards(*filter=None*, *startAt=0*, *maxResults=20*) → *ResultList*[*Dashboard*]

Return a ResultList of Dashboard resources and a `total` count.

Parameters

- **filter** (*Optional[str]*) – either "favourite" or "my", the type of dashboards to return
- **startAt** (*int*) – index of the first dashboard to return (Default: `0`)
- **maxResults** (*int*) – maximum number of dashboards to return. If maxResults set to False, it will try to get all items in batches. (Default: `20`)

Returns
ResultList[Dashboard]

dashboard(*id: str*) → *Dashboard*

Get a dashboard Resource from the server.

Parameters
id (*str*) – ID of the dashboard to get.

Returns
Dashboard

create_dashboard(***kwargs*)

copy_dashboard(***kwargs*)

update_dashboard_automatic_refresh_minutes(***kwargs*)

dashboard_item_property_keys(*dashboard_id: str*, *item_id: str*) → *ResultList*[*DashboardItemPropertyKey*]

Return a ResultList of a Dashboard gadget's property keys.

Parameters

- **dashboard_id** (*str*) – ID of dashboard.
- **item_id** (*str*) – ID of dashboard item (`DashboardGadget`).

Returns
ResultList[DashboardItemPropertyKey]

dashboard_item_property(*dashboard_id: str*, *item_id: str*, *property_key: str*) → *DashboardItemProperty*

Get the item property for a specific dashboard item (DashboardGadget).

Parameters

- **dashboard_id** (*str*) – of the dashboard.
- **item_id** (*str*) – ID of the item (`DashboardGadget`) on the dashboard.
- **property_key** (*str*) – KEY of the gadget property.

Returns
DashboardItemProperty

set_dashboard_item_property(*dashboard_id: str*, *item_id: str*, *property_key: str*, *value: dict[str, Any]*) → *DashboardItemProperty*

Set a dashboard item property.

Parameters

- **dashboard_id** (*str*) – Dashboard id.
- **item_id** (*str*) – ID of dashboard item (`DashboardGadget`) to add property_key to.
- **property_key** (*str*) – The key of the property to set.
- **value** (*dict[str, Any]*) – The dictionary containing the value of the property key.

Returns
DashboardItemProperty

dashboard_gadgets(***kwargs*)

all_dashboard_gadgets(***kwargs*)

add_gadget_to_dashboard(***kwargs*)

fields() → list[dict[str, Any]]

Return a list of all issue fields.

Returns
List[Dict[str, Any]]

filter(*id: str*) → *Filter*

Get a filter Resource from the server.

Parameters
id (*str*) – ID of the filter to get.

Returns
Filter

favourite_filters() → list[*jira.resources.Filter*]

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

Returns
List[Filter]

create_filter(*name: str = None*, *description: str = None*, *jql: str = None*, *favourite: bool = None*) → *Filter*

Create a new filter and return a filter Resource for it.

Parameters

- **name** (*str*) – name of the new filter
- **description** (*str*) – Useful human-readable description of the new filter
- **jql** (*str*) – query string that defines the filter
- **favourite** (*Optional[bool]*) – True adds this filter to the current user's favorites (Default: `None`)

Returns
Filter

update_filter(*filter_id*, *name: str = None*, *description: str = None*, *jql: str = None*, *favourite: bool = None*)

Update a filter and return a filter Resource for it.

Parameters

- **name** (*Optional[str]*) – name of the new filter

- **description** (*Optional[str]*) – Useful human-readable description of the new filter
- **jql** (*Optional[str]*) – query string that defines the filter
- **favourite** (*Optional[bool]*) – True to add this filter to the current user's favorites (Default: None)

group(*id: str*, *expand: Any = None*) → *Group*

Get a group Resource from the server.

Parameters

- **id** (*str*) – ID of the group to get
- **expand** (*Optional[Any]*) – Extra information to fetch inside each resource

Returns

Group

groups(*query: str | None = None*, *exclude: Any | None = None*, *maxResults: int = 9999*) → list[str]

Return a list of groups matching the specified criteria.

Parameters

- **query** (*Optional[str]*) – filter groups by name with this string
- **exclude** (*Optional[Any]*) – filter out groups by name with this string
- **maxResults** (*int*) – maximum results to return. (Default: 9999)

Returns

List[str]

group_members(*group: str*) → OrderedDict

Return a hash or users with their information. Requires Jira 6.0 or will raise NotImplemented.

Parameters

group (*str*) – Name of the group.

add_group(*groupname: str*) → bool

Create a new group in Jira.

Parameters

groupname (*str*) – The name of the group you wish to create.

Returns

bool – True if successful.

remove_group(*groupname: str*) → bool

Delete a group from the Jira instance.

Parameters

groupname (*str*) – The group to be deleted from the Jira instance.

Returns

bool – Returns True on success.

issue(*id: Issue | str*, *fields: str | None = None*, *expand: str | None = None*, *properties: str | None = None*) → *Issue*

Get an issue Resource from the server.

Parameters

- **id** (*Union[Issue, str]*) – ID or key of the issue to get
- **fields** (*Optional[str]*) – comma-separated string of issue fields to include in the results
- **expand** (*Optional[str]*) – extra information to fetch inside each resource

- **properties** (*Optional[str]*) – extra properties to fetch inside each result

Returns
Issue

create_issue(*fields: dict[str, Any] | None = None*, *prefetch: bool = True*, ***fieldargs*) → *Issue*

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' set of methods. Further examples are available here: https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+Example+-+Create+Issue

Parameters

- **fields** (*Optional[Dict[str, Any]]*) – a dict containing field names and the values to use. If present, all other keyword arguments will be ignored
- **prefetch** (*bool*) – True reloads the created issue Resource so all of its data is present in the value returned (Default: `True`)

Returns
Issue

create_issues(*field_list: list[dict[str, Any]]*, *prefetch: bool = True*) → list[dict[str, Any]]

Bulk create new issues and return an issue Resource for each successfully created issue.

See *create_issue* documentation for field information.

Parameters

- **field_list** (*List[Dict[str, Any]]*) – 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** (*bool*) – True reloads the created issue Resource so all of its data is present in the value returned (Default: `True`)

Returns
List[Dict[str, Any]]

supports_service_desk()

Returns if the Jira instance supports service desk.

Returns
bool

create_customer(*email: str*, *displayName: str*) → *Customer*

Create a new customer and return an issue Resource for it.

Parameters

- **email** (*str*) – Customer Email
- **displayName** (*str*) – Customer display name

Returns
Customer

service_desks() → list[*jira.resources.ServiceDesk*]

Get a list of ServiceDesk Resources from the server visible to the current authenticated user.

Returns
List[ServiceDesk]

service_desk(*id: str*) → *ServiceDesk*

Get a Service Desk Resource from the server.

Parameters
id (*str*) – ID or key of the Service Desk to get

Returns
ServiceDesk

create_customer_request(*fields: dict[str, Any] = None*, *prefetch: bool = True*, ***fieldargs*) → *Issue*

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

Parameters

- **fields** (*Dict[str, Any]*) – a dict containing field names and the values to use. If present, all other keyword arguments will be ignored
- **prefetch** (*bool*) – True reloads the created issue Resource so all of its data is present in the value returned (Default: True)

Returns
Issue

createmeta_issuetypes(*projectIdOrKey: str | int*, *startAt: int = 0*, *maxResults: int = 50*) → dict[str, Any]

Get the issue types metadata for a given project, required to create issues.

Deprecated since version 3.6.0: Use *project_issue_types()* instead.

This API was introduced in JIRA Server / DC 8.4 as a replacement for the more general purpose API 'createmeta'. For details see: https://confluence.atlassian.com/jiracore/createmeta-rest-endpoint-to-be-removed-975040986.html

Parameters

- **projectIdOrKey** (*Union[str, int]*) – id or key of the project for which to get the metadata.
- **startAt** (*int*) – Index of the first issue to return. (Default: 0)
- **maxResults** (*int*) – Maximum number of issues to return. Total number of results is available in the total attribute of the returned *ResultList*. If maxResults evaluates to False, it will try to get all issues in batches. (Default: 50)

Returns
Dict[str, Any]

createmeta_fieldtypes(*projectIdOrKey: str | int*, *issueTypeId: str | int*, *startAt: int = 0*, *maxResults: int = 50*) → dict[str, Any]

Get the field metadata for a given project and issue type, required to create issues.

Deprecated since version 3.6.0: Use *project_issue_fields()* instead.

This API was introduced in JIRA Server / DC 8.4 as a replacement for the more general purpose API 'createmeta'. For details see: https://confluence.atlassian.com/jiracore/createmeta-rest-endpoint-to-be-removed-975040986.html

Parameters

- **projectIdOrKey** (*Union[str, int]*) – id or key of the project for which to get the metadata.
- **issueTypeId** (*Union[str, int]*) – id of the issue type for which to get the metadata.
- **startAt** (*int*) – Index of the first issue to return. (Default: 0)
- **maxResults** (*int*) – Maximum number of issues to return. Total number of results is available in the `total` attribute of the returned *ResultList*. If maxResults evaluates to False, it will try to get all issues in batches. (Default: 50)

Returns

Dict[str, Any]

createmeta(*projectKeys: tuple[str, str] | str | None = None*, *projectIds: list | tuple[str, str] = []*, *issuetypeIds: list[str] | None = None*, *issuetypeNames: str | None = None*, *expand: str | None = None*) → dict[str, Any]

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

Parameters

- **projectKeys** (*Optional[Union[Tuple[str, str], str]]*) – 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** (*Union[List, Tuple[str, str]]*) – 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** (*Optional[List[str]]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – extra information to fetch inside each resource.

Returns

Dict[str, Any]

assign_issue(*issue: int | str*, *assignee: str | None*) → bool

Assign an issue to a user.

Parameters

- **issue** (*Union[int, str]*) – the issue ID or key to assign
- **assignee** (*str*) – the user to assign the issue to. None will set it to unassigned. -1 will set it to Automatic.

Returns

bool

comments(*issue: int | str*, *expand: str | None = None*) → list[*jira.resources.Comment*]

Get a list of comment Resources of the issue provided.

Parameters

- **issue** (*Union[int, str]*) – the issue ID or key to get the comments from
- **expand** (*Optional[str]*) – extra information to fetch for each comment such as renderedBody and properties.

Returns

List[Comment]

comment(*issue: int | str*, *comment: str*, *expand: str | None = None*) → *Comment*

Get a comment Resource from the server for the specified ID.

Parameters

- **issue** (*Union[int, str]*) – the issue ID or key to get the comment from
- **comment** (*str*) – ID of the comment to get
- **expand** (*Optional[str]*) – extra information to fetch for each comment such as renderedBody and properties.

Returns

Comment

add_comment(*issue: str | int | Issue*, *body: str*, *visibility: dict[str, str] | None = None*, *is_internal: bool = False*) → *Comment*

Add a comment from the current authenticated user on the specified issue and return a Resource for it.

Parameters

- **issue** (*Union[str, int, jira.resources.Issue]*) – ID or key of the issue to add the comment to
- **body** (*str*) – Text of the comment to add
- **visibility** (*Optional[Dict[str, str]]*) – a dict containing two entries: "type" and "value". "type" is 'role' (or 'group' if the Jira server has configured comment visibility for groups) "value" is the name of the role (or group) to which viewing of this comment will be restricted.
- **is_internal** (*bool*) – True marks the comment as 'Internal' in Jira Service Desk (Default: `False`)

Returns

Comment – the created comment

editmeta(*issue: str | int*)

Get the edit metadata for an issue.

Parameters

issue (*Union[str, int]*) – the issue to get metadata for

Returns

Dict[str, Dict[str, Dict[str, Any]]]

remote_links(*issue: str | int*) → list[*jira.resources.RemoteLink*]

Get a list of remote link Resources from an issue.

Parameters

issue (*Union[str, int]*) – the issue to get remote links from

Returns

List[RemoteLink]

remote_link(*issue: str | int*, *id: str*) → *RemoteLink*

Get a remote link Resource from the server.

Parameters

- **issue** (*Union[str, int]*) – the issue holding the remote link
- **id** (*str*) – ID of the remote link

Returns
RemoteLink

add_remote_link(*issue: str*, *destination: Issue | dict[str, Any]*, *globalId: str | None = None*, *application: dict[str, Any] | None = None*, *relationship: str | None = None*) → *RemoteLink*

Add a remote link from an issue to an external application and returns a remote link Resource for it.

`destination` 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 `destination` and the keyword arguments `globalId`, `application` and `relationship`, see https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+for+Remote+Issue+Links.

Parameters

- **issue** (*str*) – the issue to add the remote link to
- **destination** (*Union[Issue, Dict[str, Any]]*) – the link details to add (see the above link for details)
- **globalId** (*Optional[str]*) – unique ID for the link (see the above link for details)
- **application** (*Optional[Dict[str,Any]]*) – application information for the link (see the above link for details)
- **relationship** (*Optional[str]*) – relationship description for the link (see the above link for details)

Returns
RemoteLink – the added remote link

add_simple_link(*issue: str*, *object: dict[str, Any]*)

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** (*str*) – the issue to add the remote link to
- **object** (*Dict[str,Any]*) – the dictionary used to create remotelink data

Returns
RemoteLink

transitions(*issue: str | int | Issue*, *id: str | None = None*, *expand=None*)

Get a list of the transitions available on the specified issue to the current user.

Parameters

- **issue** (*Union[str, int, jira.resources.Issue]*) – ID or key of the issue to get the transitions from

- **id** (*Optional[str]*) – if present, get only the transition matching this ID
- **expand** (*Optional*) – extra information to fetch inside each transition

Returns
Any – json of response

find_transitionid_by_name(*issue: str | int | Issue*, *transition_name: str*) → int | None

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** (*Union[str, int, jira.resources.Issue]*) – ID or key of the issue to get the transitions from
- **transition_name** (*str*) – name of transition we are looking for

Returns
Optional[int] – returns the id is found None when it's not

transition_issue(*issue: str | int | Issue*, *transition: str*, *fields: dict[str, Any] | None = None*, *comment: str | None = None*, *worklog: str | None = 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** (*Union[str, int, jira.resources.Issue]*) – ID or key of the issue to perform the transition on
- **transition** (*str*) – ID or name of the transition to perform
- **fields** (*Optional[Dict[str,Any]]*) – a dict containing field names and the values to use.
- **comment** (*Optional[str]*) – String to add as comment to the issue when performing the transition.
- **worklog** (*Optional[str]*) – String to add as time spent on the issue when performing the transition.
- ****fieldargs** – If present, all other keyword arguments will be ignored

votes(*issue: str | int*) → *Votes*

Get a votes Resource from the server.

Parameters
issue (*Union[str, int]*) – ID or key of the issue to get the votes for

Returns
Votes

project_issue_security_level_scheme(*project: str*) → *IssueSecurityLevelScheme*

Get a IssueSecurityLevelScheme Resource from the server.

Parameters
project (*str*) – ID or key of the project to get the IssueSecurityLevelScheme for

Returns
IssueSecurityLevelScheme – The issue security level scheme

project_notification_scheme(*project: str*) → *NotificationScheme*

Get a NotificationScheme Resource from the server.

Parameters
project (*str*) – ID or key of the project to get the NotificationScheme for

Returns
NotificationScheme – The notification scheme

project_permissionscheme(*project: str*) → *PermissionScheme*

Get a PermissionScheme Resource from the server.

Parameters
project (*str*) – ID or key of the project to get the permissionscheme for

Returns
PermissionScheme – The permission scheme

project_priority_scheme(*project: str*) → *PriorityScheme*

Get a PriorityScheme Resource from the server.

Parameters
project (*str*) – ID or key of the project to get the PriorityScheme for

Returns
PriorityScheme – The priority scheme

project_workflow_scheme(*project: str*) → *WorkflowScheme*

Get a WorkflowScheme Resource from the server.

Parameters
project (*str*) – ID or key of the project to get the WorkflowScheme for

Returns
WorkflowScheme – The workflow scheme

add_vote(*issue: str | int*) → Response

Register a vote for the current authenticated user on an issue.

Parameters
issue (*Union[str, int]*) – ID or key of the issue to vote on

Returns
Response

remove_vote(*issue: str | int*)

Remove the current authenticated user's vote from an issue.

Parameters
issue (*Union[str, int]*) – ID or key of the issue to remove vote on

watchers(*issue: str | int*) → *Watchers*

Get a watchers Resource from the server for an issue.

Parameters
issue (*Union[str, int]*) – ID or key of the issue to get the watchers for

Returns
Watchers

add_watcher(*issue: str | int*, *watcher: str*) → Response

Add a user to an issue's watchers list.

Parameters

- **issue** (*Union[str, int]*) – ID or key of the issue affected
- **watcher** (*str*) – name of the user to add to the watchers list

Returns
Response

remove_watcher(*issue: str | int*, *watcher: str*) → Response

Remove a user from an issue's watch list.

Parameters

- **issue** (*Union[str, int]*) – ID or key of the issue affected
- **watcher** (*str*) – name of the user to remove from the watchers list

Returns
Response

worklogs(*issue: str | int*) → list[*jira.resources.Worklog*]

Get a list of worklog Resources from the server for an issue.

Parameters
issue (*Union[str, int]*) – ID or key of the issue to get worklogs from

Returns
List[Worklog]

worklog(*issue: str | int*, *id: str*) → *Worklog*

Get a specific worklog Resource from the server.

Parameters

- **issue** (*Union[str, int]*) – ID or key of the issue to get the worklog from
- **id** (*str*) – ID of the worklog to get

Returns
Worklog

add_worklog(*issue: str | int*, *timeSpent: str | None = None*, *timeSpentSeconds: str | None = None*, *adjustEstimate: str | None = None*, *newEstimate: str | None = None*, *reduceBy: str | None = None*, *comment: str | None = None*, *started: datetime | None = None*, *user: str | None = None*, *visibility: dict[str, Any] | None = None*) → *Worklog*

Add a new worklog entry on an issue and return a Resource for it.

Parameters

- **issue** (*Union[str, int]*) – the issue to add the worklog to
- **timeSpent** (*Optional[str]*) – a worklog entry with this amount of time spent, e.g. "2d"
- **timeSpentSeconds** (*Optional[str]*) – a worklog entry with this amount of time spent in seconds
- **adjustEstimate** (*Optional[str]*) – 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** (*Optional[str]*) – the new value for the remaining estimate field. e.g. "2d"
- **reduceBy** (*Optional[str]*) – the amount to reduce the remaining estimate by e.g. "2d"
- **comment** (*Optional[str]*) – optional worklog comment
- **started** (*Optional[datetime.datetime]*) – Moment when the work is logged, if not specified will default to now
- **user** (*Optional[str]*) – the user ID or name to use for this worklog

- **visibility** (*Optional[Dict[str,Any]]*) – Details about any restrictions in the visibility of the worklog. Optional when creating or updating a worklog.

```
```js
{
    "type": "group", # "group" or "role"
    "value": "<string>",
    "identifier": "<string>" # OPTIONAL
}
```
```

Returns
Worklog

issue_properties(*issue: str*) → list[*jira.resources.IssueProperty*]

Get a list of issue property Resource from the server for an issue.

Parameters
issue (*str*) – ID or key of the issue to get properties from

Returns
List[IssueProperty]

issue_property(*issue: str*, *key: str*) → *IssueProperty*

Get a specific issue property Resource from the server.

Parameters

- **issue** (*str*) – ID or key of the issue to get the property from
- **key** (*str*) – Key of the property to get

Returns
IssueProperty

add_issue_property(*issue: str*, *key: str*, *data*) → Response

Add or update a specific issue property Resource.

Parameters

- **issue** (*str*) – ID or key of the issue to set the property to
- **key** (*str*) – Key of the property to set
- **data** – The data to set for the property

Returns
Response

create_issue_link(*type: str | IssueLinkType*, *inwardIssue: str*, *outwardIssue: str*, *comment: dict[str, Any] | None = None*) → Response

Create a link between two issues.

Parameters

- **type** (*Union[str,IssueLinkType]*) – the type of link to create
- **inwardIssue** – the issue to link from
- **outwardIssue** – the issue to link to
- **comment** (*Optional[Dict[str, Any]]*) – 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.

Returns
Response

`delete_issue_link`(*id: str*)

Delete a link between two issues.

Parameters
id (*str*) – ID of the issue link to delete

`issue_link`(*id: str*) → *IssueLink*

Get an issue link Resource from the server.

Parameters
id (*str*) – ID of the issue link to get

Returns
IssueLink

`issue_link_types`(*force: bool = False*) → list[*jira.resources.IssueLinkType*]

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

Parameters
force (*bool*) – True forces an update of the cached IssueLinkTypes. (Default: `False`)

Returns
List[IssueLinkType]

`issue_link_type`(*id: str*) → *IssueLinkType*

Get an issue link type Resource from the server.

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

Returns
IssueLinkType

`issue_types`() → list[*jira.resources.IssueType*]

Get a list of issue type Resources from the server.

Returns
List[IssueType]

`project_issue_types`(*project: str*, *startAt: int = 0*, *maxResults: int = 50*) → *ResultList*[*IssueType*]

Get a list of issue type Resources available in a given project from the server.

This API was introduced in JIRA Server / DC 8.4 as a replacement for the more general purpose API 'createmeta'. For details see: https://confluence.atlassian.com/jiracore/createmeta-rest-endpoint-to-be-removed-975040986.html

Parameters

- **project** (*str*) – ID or key of the project to query issue types from.
- **startAt** (*int*) – Index of first issue type to return. (Default: `0`)
- **maxResults** (*int*) – Maximum number of issue types to return. (Default: `50`)

Returns
ResultList[IssueType]

`project_issue_fields`(*project: str*, *issue_type: str*, *startAt: int = 0*, *maxResults: int = 50*) → *ResultList*[Field]

Get a list of field type Resources available for a project and issue type from the server.

This API was introduced in JIRA Server / DC 8.4 as a replacement for the more general purpose API 'createmeta'. For details see: https://confluence.atlassian.com/jiracore/createmeta-rest-endpoint-to-be-removed-975040986.html

Parameters

- **project** (*str*) – ID or key of the project to query field types from.
- **issue_type** (*str*) – ID of the issue type to query field types from.
- **startAt** (*int*) – Index of first issue type to return. (Default: 0)
- **maxResults** (*int*) – Maximum number of issue types to return. (Default: 50)

Returns
ResultList[Field]

issue_type(*id: str*) → *IssueType*

Get an issue type Resource from the server.

Parameters
id (*str*) – ID of the issue type to get

Returns
IssueType

issue_type_by_name(*name: str*, *project: str | None = None*) → *IssueType*

Get issue type by name.

Parameters

- **name** (*str*) – Name of the issue type
- **project** (*str*) – Key or ID of the project. If set, only issue types available for that project will be looked up.

Returns
IssueType

request_types(*service_desk:* ServiceDesk) → list[*jira.resources.RequestType*]

Returns request types supported by a service desk instance.

Parameters
service_desk (ServiceDesk) – The service desk instance.

Returns
List[RequestType]

request_type_by_name(*service_desk:* ServiceDesk, *name: str*)

my_permissions(*projectKey: str | None = None*, *projectId: str | None = None*, *issueKey: str | None = None*, *issueId: str | None = None*, *permissions: str | None = None*) → dict[str, dict[str, dict[str, str]]]

Get a dict of all available permissions on the server.

`permissions` is a comma-separated value list of permission keys that is required in Jira Cloud. For possible and allowable permission values, see https://developer.atlassian.com/cloud/jira/platform/rest/v3/api-group-permission-schemes/#built-in-permissions

Parameters

- **projectKey** (*Optional[str]*) – limit returned permissions to the specified project
- **projectId** (*Optional[str]*) – limit returned permissions to the specified project
- **issueKey** (*Optional[str]*) – limit returned permissions to the specified issue
- **issueId** (*Optional[str]*) – limit returned permissions to the specified issue
- **permissions** (*Optional[str]*) – limit returned permissions to the specified csv permission keys (cloud required field)

Returns
Dict[str, Dict[str, Dict[str, str]]]

priorities() → list[*jira.resources.Priority*]

Get a list of priority Resources from the server.

Returns
List[Priority]

priority(*id: str*) → *Priority*

Get a priority Resource from the server.

Parameters
id (*str*) – ID of the priority to get

Returns
Priority

projects(*expand: str | None = None*) → list[*jira.resources.Project*]

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

Parameters
expand (*Optional[str]*) – extra information to fetch for each project such as projectKeys and description.

Returns
List[Project]

project(*id: str*, *expand: str | None = None*) → *Project*

Get a project Resource from the server.

Parameters

- **id** (*str*) – ID or key of the project to get
- **expand** (*Optional[str]*) – extra information to fetch for the project such as projectKeys and description.

Returns
Project

project_avatars(*project: str*)

Get a dict of all avatars for a project visible to the current authenticated user.

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

create_temp_project_avatar(*project: str*, *filename: str*, *size: int*, *avatar_img: bytes*, *contentType: str = None*, *auto_confirm: bool = 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 Their Documentation 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** (*str*) – ID or key of the project to create the avatar in
- **filename** (*str*) – name of the avatar file
- **size** (*int*) – size of the avatar file
- **avatar_img** (*bytes*) – file-like object holding the avatar
- **contentType** (*str*) – explicit specification for the avatar image's content-type
- **auto_confirm** (*bool*) – True to automatically confirm the temporary avatar by calling *confirm_project_avatar()* with the return value of this method. (Default: `False`)

confirm_project_avatar(*project: str*, *cropping_properties: dict[str, Any]*)

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** (*str*) – ID or key of the project to confirm the avatar in
- **cropping_properties** (*Dict[str,Any]*) – a dict of cropping properties from *create_temp_project_avatar()*

set_project_avatar(*project: str*, *avatar: str*)

Set a project's avatar.

Parameters

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

delete_project_avatar(*project: str*, *avatar: str*) → Response

Delete a project's avatar.

Parameters

- **project** (*str*) – ID or key of the project to delete the avatar from
- **avatar** (*str*) – ID of the avatar to delete

Returns
Response

project_components(*project: str*) → list[*jira.resources.Component*]

Get a list of component Resources present on a project.

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

Returns
List[Component]

project_versions(*project: str*) → list[*jira.resources.Version*]

Get a list of version Resources present on a project.

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

Returns
List[Version]

get_project_version_by_name(*project: str*, *version_name: str*) → *Version* | None

Get a version Resource by its name present on a project.

Parameters

- **project** (*str*) – ID or key of the project to get versions from
- **version_name** (*str*) – name of the version to search for

Returns
Optional[Version]

rename_version(*project: str*, *old_name: str*, *new_name: str*) → None

Rename a version Resource on a project.

Parameters

- **project** (*str*) – ID or key of the project to get versions from
- **old_name** (*str*) – old name of the version to rename
- **new_name** (*str*) – new name of the version to rename

project_roles(*project: str*) → dict[str, dict[str, str]]

Get a dict of role names to resource locations for a project.

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

Returns
Dict[str, Dict[str, str]]

project_role(*project: str*, *id: str*) → *Role*

Get a role Resource.

Parameters

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

Returns
Role

resolutions() → list[*jira.resources.Resolution*]

Get a list of resolution Resources from the server.

Returns
List[Resolution]

resolution(*id: str*) → *Resolution*

Get a resolution Resource from the server.

Parameters
id (*str*) – ID of the resolution to get

Returns
Resolution

search_issues(*jql_str: str*, *startAt: int = 0*, *maxResults: int = 50*, *validate_query: bool = True*, *fields: str | list[str] | None = '*all'*, *expand: str | None = None*, *properties: str | None = None*, *json_result: bool = False*, *use_post: bool = False*) → dict[str, Any] | *ResultList*[*Issue*]

Get a *ResultList* of issue Resources matching a JQL search string.

Parameters

- **jql_str** (*str*) – The JQL search string.
- **startAt** (*int*) – Index of the first issue to return. (Default: 0)

- **maxResults** (*int*) – Maximum number of issues to return. Total number of results is available in the `total` attribute of the returned *ResultList*. If maxResults evaluates to False, it will try to get all issues in batches. (Default: `50`)
- **validate_query** (*bool*) – True to validate the query. (Default: `True`)
- **fields** (*Optional[Union[str, List[str]]]*) – comma-separated string or list of issue fields to include in the results. Default is to include all fields.
- **expand** (*Optional[str]*) – extra information to fetch inside each resource
- **properties** (*Optional[str]*) – extra properties to fetch inside each result
- **json_result** (*bool*) – True to return a JSON response. When set to False a *ResultList* will be returned. (Default: `False`)
- **use_post** (*bool*) – True to use POST endpoint to fetch issues.

Returns
Union[Dict,ResultList] – Dict if `json_result=True`

security_level(*id: str*) → *SecurityLevel*

Get a security level Resource.

Parameters
id (*str*) – ID of the security level to get

Returns
SecurityLevel

server_info() → dict[str, Any]

Get a dict of server information for this Jira instance.

Returns
Dict[str, Any]

myself() → dict[str, Any]

Get a dict of server information for this Jira instance.

Returns
Dict[str, Any]

statuses() → list[*jira.resources.Status*]

Get a list of all status Resources from the server.

Refer to *JIRA.issue_types_for_project()* for getting statuses for a specific issue type within a specific project.

Returns
List[Status]

issue_types_for_project(*projectIdOrKey: str*) → list[*jira.resources.IssueType*]

Get a list of issue types available within the project.

Each project has a set of valid issue types and each issue type has a set of valid statuses. The valid statuses for a given issue type can be extracted via: *issue_type_x.statuses*

Returns
List[IssueType]

status(*id: str*) → *Status*

Get a status Resource from the server.

Parameters
id (*str*) – ID of the status resource to get

Returns
Status

statuscategories() → list[*jira.resources.StatusCategory*]

Get a list of status category Resources from the server.

Returns
List[StatusCategory]

statuscategory(*id: int*) → *StatusCategory*

Get a status category Resource from the server.

Parameters
id (*int*) – ID of the status category resource to get

Returns
StatusCategory

user(*id: str*, *expand: Any | None = None*) → *User*

Get a user Resource from the server.

Parameters

- **id** (*str*) – ID of the user to get
- **expand** (*Optional[Any]*) – Extra information to fetch inside each resource

Returns
User

search_assignable_users_for_projects(*username: str*, *projectKeys: str*, *startAt: int = 0*, *maxResults: int = 50*) → *ResultList*

Get a list of user Resources that match the search string and can be assigned issues for projects.

Parameters

- **username** (*str*) – A string to match usernames against
- **projectKeys** (*str*) – Comma-separated list of project keys to check for issue assignment permissions
- **startAt** (*int*) – Index of the first user to return (Default: 0)
- **maxResults** (*int*) – Maximum number of users to return. If maxResults evaluates as False, it will try to get all users in batches. (Default: 50)

Returns
ResultList

search_assignable_users_for_issues(*username: str | None = None*, *project: str | None = None*, *issueKey: str | None = None*, *expand: Any | None = None*, *startAt: int = 0*, *maxResults: int = 50*, *query: str | None = None*)

Get a list of user Resources that match the search string for assigning or creating issues.

"username" query parameter is deprecated in Jira Cloud; the expected parameter now is "query", which can just be the full email again. But the "user" parameter is kept for backwards compatibility, i.e. Jira Server/Data Center.

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** (*Optional[str]*) – A string to match usernames against
- **project** (*Optional[str]*) – Filter returned users by permission in this project (expected if a result will be used to create an issue)

- **issueKey** (*Optional[str]*) – Filter returned users by this issue (expected if a result will be used to edit this issue)
- **expand** (*Optional[Any]*) – Extra information to fetch inside each resource
- **startAt** (*int*) – Index of the first user to return (Default: 0)
- **maxResults** (*int*) – maximum number of users to return. If maxResults evaluates as False, it will try to get all items in batches. (Default: 50)
- **query** (*Optional[str]*) – Search term. It can just be the email.

Returns
ResultList

user_avatars(*username: str*) → dict[str, Any]

Get a dict of avatars for the specified user.

Parameters
username (*str*) – the username to get avatars for

Returns
Dict[str, Any]

create_temp_user_avatar(*user: str*, *filename: str*, *size: int*, *avatar_img: bytes*, *contentType: Any = None*, *auto_confirm: bool = 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 Their Documentation 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** (*str*) – User to register the avatar for
- **filename** (*str*) – name of the avatar file
- **size** (*int*) – size of the avatar file
- **avatar_img** (*bytes*) – file-like object containing the avatar
- **contentType** (*Optional[Any]*) – explicit specification for the avatar image's content-type
- **auto_confirm** (*bool*) – True to automatically confirm the temporary avatar by calling *confirm_user_avatar()* with the return value of this method. (Default: `False`)

confirm_user_avatar(*user: str*, *cropping_properties: dict[str, Any]*)

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** (*str*) – the user to confirm the avatar for
- **cropping_properties** (*Dict[str,Any]*) – a dict of cropping properties from *create_temp_user_avatar()*

set_user_avatar(*username: str*, *avatar: str*) → Response

Set a user's avatar.

Parameters

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

Returns
Response

delete_user_avatar(*username: str*, *avatar: str*) → Response

Delete a user's avatar.

Parameters

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

Returns
Response

delete_remote_link(*issue: str | Issue*, ***, *internal_id: str | None = None*, *global_id: str | None = None*) → Response

Delete remote link from issue by internalId or globalId.

Parameters

- **issue** (*str*) – Key (or Issue) of Issue
- **internal_id** (*Optional[str]*) – InternalID of the remote link to delete
- **global_id** (*Optional[str]*) – GlobalID of the remote link to delete

Returns
Response

search_users(*user: str | None = None*, *startAt: int = 0*, *maxResults: int = 50*, *includeActive: bool = True*, *includeInactive: bool = False*, *query: str | None = None*) → *ResultList*[*User*]

Get a list of user Resources that match the specified search string.

"username" query parameter is deprecated in Jira Cloud; the expected parameter now is "query", which can just be the full email again. But the "user" parameter is kept for backwards compatibility, i.e. Jira Server/Data Center.

Parameters

- **user** (*Optional[str]*) – a string to match usernames, name or email against.
- **startAt** (*int*) – index of the first user to return.
- **maxResults** (*int*) – maximum number of users to return. If maxResults evaluates as False, it will try to get all items in batches.
- **includeActive** (*bool*) – True to include active users in the results. (Default: `True`)
- **includeInactive** (*bool*) – True to include inactive users in the results. (Default: `False`)
- **query** (*Optional[str]*) – Search term. It can just be the email.

Returns
ResultList[User]

search_allowed_users_for_issue(*user: str*, *issueKey: str = None*, *projectKey: str = None*, *startAt: int = 0*, *maxResults: int = 50*) → *ResultList*

Get a list of user Resources that match a username string and have browse permission for the issue or project.

Parameters

- **user** (*str*) – a string to match usernames against.
- **issueKey** (*Optional[str]*) – find users with browse permission for this issue.
- **projectKey** (*Optional[str]*) – find users with browse permission for this project.
- **startAt** (*int*) – index of the first user to return. (Default: `0`)
- **maxResults** (*int*) – maximum number of users to return. If maxResults evaluates as False, it will try to get all items in batches. (Default: `50`)

Returns

ResultList

create_version(*name: str*, *project: str*, *description: str = None*, *releaseDate: Any = None*, *startDate: Any = None*, *archived: bool = False*, *released: bool = False*) → *Version*

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

Parameters

- **name** (*str*) – name of the version to create
- **project** (*str*) – key of the project to create the version in
- **description** (*str*) – a description of the version
- **releaseDate** (*Optional[Any]*) – the release date assigned to the version
- **startDate** (*Optional[Any]*) – The start date for the version
- **archived** (*bool*) – True to create an archived version. (Default: `False`)
- **released** (*bool*) – True to create a released version. (Default: `False`)

Returns

Version

move_version(*id: str*, *after: str = None*, *position: str = None*) → *Version*

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

Returns

Version

version(*id: str*, *expand: Any = None*) → *Version*

Get a version Resource.

Parameters

- **id** (*str*) – ID of the version to get
- **expand** (*Optional[Any]*) – extra information to fetch inside each resource

Returns
Version

`version_count_related_issues`(*id: str*)

Get a dict of the counts of issues fixed and affected by a version.

Parameters
id (*str*) – the version to count issues for

`version_count_unresolved_issues`(*id: str*)

Get the number of unresolved issues for a version.

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

`session`() → *User*

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

Returns
User

`kill_session`() → Response

Destroy the session of the current authenticated user.

Returns
Response

`kill_websudo`() → Response | None

Destroy the user's current WebSudo session.

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

Returns
Optional[Response]

`rename_user`(*old_user: str*, *new_user: str*)

Rename a Jira user.

Parameters

- **old_user** (*str*) – Old username login
- **new_user** (*str*) – New username login

`delete_user`(*username: str*) → bool

Deletes a Jira User.

Parameters
username (*str*) – Username to delete

Returns
bool – Success of user deletion

`deactivate_user`(*username: str*) → str | int

Disable/deactivate the user.

Parameters
username (*str*) – User to be deactivated.

Returns
Union[str, int]

`reindex`(*force: bool = False*, *background: bool = True*) → bool

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 background reindex only if Jira thinks it should do it.

Parameters

- **force** (*bool*) – True to reindex even if Jira doesn't say this is needed. (Default: `False`)
- **background** (*bool*) – True to reindex in background, slower but does not impact the users. (Default: `True`)

Returns

bool – True if reindexing is in progress or not needed

backup(*filename: str = 'backup.zip'*, *attachments: bool = False*) → bool | int | None

Will call jira export to backup as zipped xml. Returning with success does not mean that the backup process finished.

Parameters

- **filename** (*str*) – the filename for the backup (Default: "backup.zip")
- **attachments** (*bool*) – True to also backup attachments (Default: `False`)

Returns

Union[bool, int] – Returns True if successful else it returns the statuscode of the Response or False

backup_progress() → dict[str, Any] | None

Return status of cloud backup as a dict.

Is there a way to get progress for Server version?

Returns

Optional[Dict[str, Any]]

backup_complete() → bool | None

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

backup_download(*filename: str = None*)

Download backup file from WebDAV (cloud only).

current_user(*field: str | None = None*) → str

Return the *accountId* (Cloud) else *username* of the current user.

For anonymous users it will return a value that evaluates as False.

Parameters

field (*Optional[str]*) – the name of the identifier field. Defaults to "accountId" for Jira Cloud, else "username"

Returns

str – User's *accountId* (Cloud) else *username*.

delete_project(*pid: str | Project*, *enable_undo: bool = True*) → bool | None

Delete project from Jira.

Parameters

- **pid** (*Union[str, Project]*) – Jira projectID or Project or slug.
- **enable_undo** (*bool*) – Jira Cloud only. True moves to 'Trash'. False permanently deletes.

Raises

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

Returns
bool – True if project was deleted

templates() → dict

permissionschemes()

issue_type_schemes() → list[*jira.resources.IssueTypeScheme*]
Get all issue type schemes defined (Admin required).

Returns
List[IssueTypeScheme] – All the Issue Type Schemes available to the currently logged in user.

issuesecurityschemes()

projectcategories()

avatars(*entity='project'*)

notificationschemes()

screens()

workflowscheme()

workflows()

delete_screen(*id: str*)

delete_permissionscheme(*id: str*)

get_issue_type_scheme_associations(*id: str*) → list[*jira.resources.Project*]
For the specified issue type scheme, returns all of the associated projects. (Admin required).

Parameters
id (*str*) – The issue type scheme id.

Returns
List[Project] – Associated Projects for the Issue Type Scheme.

create_project(*key: str*, *name: str = None*, *assignee: str = None*, *ptype: str = 'software'*, *template_name: str = None*, *avatarId: int = None*, *issueSecurityScheme: int = None*, *permissionScheme: int = None*, *projectCategory: int = None*, *notificationScheme: int = 10000*, *categoryId: int = None*, *url: str = ''*)
Create a project with the specified parameters.

Parameters

- **key** (*str*) – Mandatory. Must match Jira project key requirements, usually only 2-10 uppercase characters.
- **name** (*Optional[str]*) – If not specified it will use the key value.
- **assignee** (*Optional[str]*) – Key of the lead, if not specified it will use current user.
- **ptype** (*Optional[str]*) – Determines the type of project that should be created. Defaults to 'software'.
- **template_name** (*Optional[str]*) – 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.
- **avatarId** (*Optional[int]*) – ID of the avatar to use for the project.

- **issueSecurityScheme** (*Optional[int]*) – Determines the security scheme to use. If none provided, will fetch the scheme named 'Default' or the first scheme returned.
- **permissionScheme** (*Optional[int]*) – Determines the permission scheme to use. If none provided, will fetch the scheme named 'Default Permission Scheme' or the first scheme returned.
- **projectCategory** (*Optional[int]*) – Determines the category the project belongs to. If none provided, will fetch the one named 'Default' or the first category returned.
- **notificationScheme** (*Optional[int]*) – Determines the notification scheme to use.
- **categoryId** (*Optional[int]*) – Same as projectCategory. Can be used interchangeably.
- **url** (*Optional[str]*) – A link to information about the project, such as documentation.

Returns

Union[bool,int] – Should evaluate to False if it fails otherwise it will be the new project id.

add_user(*username: str*, *email: str*, *directoryId: int = 1*, *password: str = None*, *fullname: str = None*, *notify: bool = False*, *active: bool = True*, *ignore_existing: bool = False*, *application_keys: list | None = None*)

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 (Default: 1)
- **password** (*Optional[str]*) – Optional, the password for the new user
- **fullname** (*Optional[str]*) – Optional, the full name of the new user
- **notify** (*bool*) – True to send a notification to the new user. (Default: `False`)
- **active** (*bool*) – True to make the new user active upon creation. (Default: `True`)
- **ignore_existing** (*bool*) – True to ignore existing users. (Default: `False`)
- **application_keys** (*Optional[list]*) – Keys of products user should have access to

Raises

JIRAError – If username already exists and *ignore_existing* has not been set to *True*.

Returns

bool – Whether the user creation was successful.

add_user_to_group(*username: str*, *group: str*) → bool | dict[str, Any]

Add a user to an existing group.

Parameters

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

Returns

Union[bool,Dict[str,Any]] – json response from Jira server for success or a value that evaluates as False in case of failure.

remove_user_from_group(*username: str*, *groupname: str*) → bool

Remove a user from a group.

Parameters

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

Returns
bool

role() → list[dict[str, Any]]

Return Jira role information.

Returns
List[Dict[str,Any]] – List of current user roles

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

boards(*startAt: int = 0*, *maxResults: int = 50*, *type: str = None*, *name: str = None*, *projectKeyOrID=None*) → *ResultList*[*Board*]

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.
- **projectKeyOrID** – Filters results to boards that match the specified project key or ID.

Returns
ResultList[Board]

sprints(*board_id: int*, *extended: bool | None = None*, *startAt: int = 0*, *maxResults: int = 50*, *state: str = None*) → *ResultList*[*Sprint*]

Get a list of sprint Resources.

Parameters

- **board_id** (*int*) – the board to get sprints from
- **extended** (*bool*) – Deprecated.
- **startAt** (*int*) – the index of the first sprint to return (0 based)
- **maxResults** (*int*) – the maximum number of sprints to return
- **state** (*str*) – Filters results to sprints in specified states. Valid values: *future*, *active*, *closed*. You can define multiple states separated by commas

Returns
ResultList[Sprint] – List of sprints.

sprints_by_name(*id: str | int*, *extended: bool = False*, *state: str = None*) → dict[str, dict[str, Any]]

Get a dictionary of sprint Resources where the name of the sprint is the key.

Parameters

- **board_id** (*int*) – the board to get sprints from
- **extended** (*bool*) – Deprecated.
- **state** (*str*) – Filters results to sprints in specified states. Valid values: *future*, *active*, *closed*. You can define multiple states separated by commas

Returns
Dict[str, Dict[str, Any]] – dictionary of sprints with the sprint name as key

update_sprint(*id: str | int*, *name: str | None = None*, *startDate: Any | None = None*, *endDate: Any | None = None*, *state: str | None = None*, *goal: str | None = None*) → dict[str, Any]

Updates the sprint with the given values.

Parameters

- **id** (*Union[str, int]*) – The id of the sprint to update
- **name** (*Optional[str]*) – The name to update your sprint to
- **startDate** (*Optional[Any]*) – The start date for the sprint
- **endDate** (*Optional[Any]*) – The start date for the sprint
- **state** – (Optional[str]): The state of the sprint
- **goal** – (Optional[str]): The goal of the sprint

Returns
Dict[str, Any]

incompletedIssuesEstimateSum(*board_id: str*, *sprint_id: str*)

Return the total incompleted points this sprint.

removed_issues(*board_id: str*, *sprint_id: str*)

Return the completed issues for the sprint.

Returns
List[Issue]

removedIssuesEstimateSum(*board_id: str*, *sprint_id: str*)

Return the total incompleted points this sprint.

sprint_info(*board_id: str*, *sprint_id: str*) → dict[str, Any]

Return the information about a sprint.

Parameters

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

Returns
Dict[str, Any]

sprint(*id: int*) → *Sprint*

Return the information about a sprint.

Parameters
sprint_id (*int*) – the sprint retrieving issues from

Returns
Sprint

delete_board(*id*)

Delete an agile board.

create_board(*name: str*, *filter_id: str*, *project_ids: str = None*, *preset: str = 'scrum'*, *location_type: Literal['user', 'project'] = 'user'*, *location_id: str | None = None*) → *Board*

Create a new board for the `project_ids`.

Parameters

- **name** (*str*) – name of the Board (<255 characters).

- **filter_id** (*str*) – the Filter to use to create the Board. Note: if the user does not have the 'Create shared objects' permission and tries to create a shared board, a private board will be created instead (remember that board sharing depends on the filter sharing).
- **project_ids** (*str*) – Deprecated. See location_id.
- **preset** (*str*) – What preset/type to use for this Board, options: kanban, scrum, agility. (Default: "scrum")
- **location_type** (*str*) – the location type. Available in Cloud. (Default: "user")
- **location_id** (*Optional[str]*) – aka `projectKeyOrId`. The id of Project that the Board should be located under. Omit this for a 'user' location_type. Available in Cloud.

Returns
Board – The newly created board

create_sprint(*name: str*, *board_id: int*, *startDate: Any | None = None*, *endDate: Any | None = None*, *goal: str | None = None*) → *Sprint*

Create a new sprint for the `board_id`.

Parameters

- **name** (*str*) – Name of the sprint
- **board_id** (*int*) – Which board the sprint should be assigned.
- **startDate** (*Optional[Any]*) – Start date for the sprint.
- **endDate** (*Optional[Any]*) – End date for the sprint.
- **goal** (*Optional[str]*) – Goal for the sprint.

Returns
Sprint – The newly created Sprint

add_issues_to_sprint(*sprint_id: int*, *issue_keys: list[str]*) → Response

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** (*int*) – the sprint to add issues to
- **issue_keys** (*List[str]*) – the issues to add to the sprint

Returns
Response

add_issues_to_epic(*epic_id: str*, *issue_keys: str | list[str]*, *ignore_epics: bool = None*) → Response

Add the issues in `issue_keys` to the `epic_id`.

Issues can only exist in one Epic!

Parameters

- **epic_id** (*str*) – The ID for the epic where issues should be added.
- **issue_keys** (*Union[str, List[str]]*) – The list (or comma separated str) of issues to add to the epic
- **ignore_epics** (*bool*) – Deprecated.

Returns
Response

rank(*issue: str*, *next_issue: str | None = None*, *prev_issue: str | None = None*) → Response

Rank an issue before/after another using the default Ranking field, the one named 'Rank'.

Pass only ONE of *next_issue* or *prev_issue*.

Parameters

- **issue** (*str*) – issue key of the issue to be ranked before/after the second one.
- **next_issue** (*str*) – issue key that the first issue is to be ranked before.
- **prev_issue** (*str*) – issue key that the first issue is to be ranked after.

Returns
Response

move_to_backlog(*issue_keys: list[str]*) → Response

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

Parameters
issue_keys (*List[str]*) – the issues to move to the backlog

Raises
JIRAError – If moving issues to backlog fails

Returns
Response

6.1.2 jira.config module

Config handler.

This module allows people to keep their jira server credentials outside their script, in a configuration file that is not saved in the source control.

Also, this simplifies the scripts by not having to write the same initialization code for each script.

jira.config.**get_jira**(*profile: str | None = None*, *url: str = 'http://localhost:2990'*, *username: str = 'admin'*, *password: str = 'admin'*, *appid=None*, *autofix=False*, *verify: bool | str = True*)

Return a JIRA object by loading the connection details from the *config.ini* file.

Parameters

- **profile** (*Optional[str]*) – The name of the section from config.ini file that stores server config url/username/password
- **url** (*str*) – URL of the Jira server
- **username** (*str*) – username to use for authentication
- **password** (*str*) – password to use for authentication
- **appid** – appid
- **autofix** – autofix
- **verify** (*Union[bool, str]*) – True to indicate whether SSL certificates should be verified or str path to a CA_BUNDLE file or directory with certificates of trusted CAs. (Default: `True`)

Returns
JIRA – an instance to a JIRA object.

Raises
EnvironmentError –

Usage:

```
>>> from jira.config import get_jira
>>>
>>> jira = get_jira(profile='jira')
```

Also create a *config.ini* like this and put it in current directory, user home directory or PYTHONPATH.

```
[jira]
url=https://jira.atlassian.com
# only the `url` is mandatory
user=...
pass=...
appid=...
verify=...
```

6.1.3 jira.exceptions module

exception jira.exceptions.**JIRAError**(*text: str = None, status_code: int = None, url: str = None, request: Response = None, response: Response = None, **kwargs*)

Bases: Exception

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

__init__(*text: str = None, status_code: int = None, url: str = None, request: Response = None, response: Response = None, **kwargs*)

Creates a JIRAError.

Parameters

- **text** (*Optional[str]*) – Message for the error.
- **status_code** (*Optional[int]*) – Status code for the error.
- **url** (*Optional[str]*) – Url related to the error.
- **request** (*Optional[requests.Response]*) – Request made related to the error.
- **response** (*Optional[requests.Response]*) – Response received related to the error.
- ****kwargs** – Will be used to get request headers.

exception jira.exceptions.**NotJIRAInstanceError**(*instance: Any*)

Bases: Exception

Raised in the case an object is not a JIRA instance.

__init__(*instance: Any*)

6.1.4 jira.jirashell module

Starts an interactive Jira session in an ipython terminal.

Script arguments support changing the server and a persistent authentication over HTTP BASIC or Kerberos.

jira.jirashell.**oauth_dance**(*server*, *consumer_key*, *key_cert_data*, *print_tokens=False*, *verify=None*)

jira.jirashell.**process_config**()

jira.jirashell.**process_command_line**()

jira.jirashell.**get_config**()

jira.jirashell.**handle_basic_auth**(*auth*, *server*)

jira.jirashell.**main**()

6.1.5 jira.resilientsession module

class jira.resilientsession.**PrepareRequestForRetry**

Bases: object

This class allows for the manipulation of the Request keyword arguments before a retry.

The *prepare()* handles the processing of the Request keyword arguments.

abstract prepare(*original_request_kwargs: CaseInsensitiveDict*) → CaseInsensitiveDict

Process the Request's keyword arguments before retrying the Request.

Parameters
original_request_kwargs (CaseInsensitiveDict) – The keyword arguments of the Request.

Returns
CaseInsensitiveDict – The new keyword arguments to use in the retried Request.

class jira.resilientsession.**PassthroughRetryPrepare**

Bases: *PrepareRequestForRetry*

Returns the Request's keyword arguments unchanged, when no change needs to be made before a retry.

prepare(*original_request_kwargs: CaseInsensitiveDict*) → CaseInsensitiveDict

jira.resilientsession.**raise_on_error**(*resp: Response | None*, ***kwargs*) → TypeGuard[Response]

Handle errors from a Jira Request.

Parameters
resp (*Optional[Response]*) – Response from Jira request

Raises

- ***JIRAError*** – If Response is None
- ***JIRAError*** – for unhandled 400 status codes.

Returns
TypeGuard[Response] – True if the passed in Response is all good.

jira.resilientsession.**parse_errors**(*resp: Response*) → list[str]

Parse a Jira Error messages from the Response.

https://developer.atlassian.com/cloud/jira/platform/rest/v2/intro/#status-codes

Parameters
resp (*Response*) – The Jira API request's response.

Returns
List[str] – The error messages list parsed from the Response. An empty list if no error.

jira.resilientsession.**parse_error_msg**(*resp: Response*) → str

Parse a Jira Error messages from the Response and join them by comma.

https://developer.atlassian.com/cloud/jira/platform/rest/v2/intro/#status-codes

Parameters
resp (*Response*) – The Jira API request's response.

Returns
str – The error message parsed from the Response. An empty str if no error.

class jira.resilientsession.**ResilientSession**(*timeout=None*, *max_retries: int = 3*, *max_retry_delay: int = 60*)

Bases: Session

This class is supposed to retry requests that do return temporary errors.

`__recoverable()` handles all retry-able errors.

__init__(*timeout=None*, *max_retries: int = 3*, *max_retry_delay: int = 60*)

A Session subclass catered for the Jira API with exponential delaying retry.

Parameters

- **timeout** (*Optional[Union[Union[float, int], Tuple[float, float]]]*) – Connection/read timeout delay. Defaults to None.
- **max_retries** (*int*) – Max number of times to retry a request. Defaults to 3.
- **max_retry_delay** (*int*) – Max delay allowed between retries. Defaults to 60.

request(*method: str*, *url: str | bytes*, *_prepare_retry_class: ~jira.resilientsession.PrepareRequestForRetry = <jira.resilientsession.PassthroughRetryPrepare object>*, ***kwargs*) → Response

This is an intentional override of *Session.request()* to inject some error handling and retry logic.

Raises
Exception – Various exceptions as defined in py:method:*raise_on_error*.

Returns
Response – The response.

6.1.6 jira.resources module

jira.client.**ResourceType = alias of TypeVar('ResourceType', contravariant=True, bound=jira.resources.Resource)**

Type variable.

Usage:

```
T = TypeVar('T')  # Can be anything
A = TypeVar('A', str, bytes)  # Must be str or bytes
```

Type variables exist primarily for the benefit of static type checkers. They serve as the parameters for generic types as well as for generic function definitions. See class Generic for more information on generic types. Generic functions work as follows:

def repeat(x: T, n: int) -> List[T]:
'''Return a list containing n references to x.''' return [x]*n

def longest(x: A, y: A) -> A:
'''Return the longest of two strings.''' return x if len(x) >= len(y) else y

The latter example's signature is essentially the overloading of (str, str) -> str and (bytes, bytes) -> bytes. Also note that if the arguments are instances of some subclass of str, the return type is still plain str.

At runtime, isinstance(x, T) and issubclass(C, T) will raise TypeError.

Type variables defined with covariant=True or contravariant=True can be used to declare covariant or contravariant generic types. See PEP 484 for more details. By default generic types are invariant in all type variables.

Type variables can be introspected. e.g.:

T.__name__ == 'T' T.__constraints__ == () T.__covariant__ == False T.__contravariant__ = False A.__constraints__ == (str, bytes)

Note that only type variables defined in global scope can be pickled.

Jira resource definitions.

This module implements the Resource classes that translate JSON from Jira REST resources into usable objects.

class jira.resources.**Resource**(*resource: str*, *options: dict[str, Any]*, *session:* ResilientSession, *base_url: str = '{server}/rest/{rest_path}/{rest_api_version}/{path}'*)

Bases: `object`

Models a URL-addressable resource in the Jira REST API.

All Resource objects provide the following: `find()` – get a resource from the server and load it into the current object (though clients should use the methods in the JIRA class instead of this method directly) `update()` – changes the value of this resource on the server and returns a new resource object for it `delete()` – deletes this resource from the server `self` – the URL of this resource on the server `raw` – dict of properties parsed out of the JSON response from the server

Subclasses will implement `update()` and `delete()` as appropriate for the specific resource.

All Resources have a resource path of the form:

- `issue`
- `project/{0}`
- `issue/{0}/votes`
- `issue/{0}/comment/{1}`

where the bracketed numerals are placeholders for ID values that are filled in from the `ids` parameter to `find()`.

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

_READABLE_IDS = ('displayName', 'key', 'name', 'accountId', 'filename', 'value', 'scope', 'votes', 'id', 'mimeType', 'closed')

_HASH_IDS = ('self', 'type', 'key', 'id', 'name')

__init__(*resource: str*, *options: dict[str, Any]*, *session:* ResilientSession, *base_url: str = '{server}/rest/{rest_path}/{rest_api_version}/{path}'*)

Initializes a generic resource.

Parameters

- **resource** (*str*) – The name of the resource.
- **options** (*Dict[str,str]*) – Options for the new resource
- **session** (ResilientSession) – Session used for the resource.
- **base_url** (*Optional[str]*) – The Base Jira url.

find(*id: tuple[str, ...] | int | str*, *params: dict[str, str] | None = None*)

Finds a resource based on the input parameters.

Parameters

- **id** (*Union[Tuple[str, str], int, str]*) – id
- **params** (*Optional[Dict[str, str]]*) – params

_find_by_url(*url: str*, *params: dict[str, str] | None = None*)

Finds a resource on the specified url.

The resource is loaded with the JSON data returned by doing a request on the specified url.

Parameters

- **url** (*str*) – url
- **params** (*Optional[Dict[str, str]]*) – params

_get_url(*path: str*) → str

Gets the url for the specified path.

Parameters
path (*str*) – str

Returns
str

update(*fields: dict[str, Any] | None = None*, *async_: bool | None = None*, *jira: JIRA = None*, *notify: bool = True*, ***kwargs: Any*)

Update this resource on the server.

Keyword arguments are marshalled into a dict before being sent. If this resource doesn't support PUT, a *JIRAError* will be raised; subclasses that specialize this method will only raise errors in case of user error.

Parameters

- **fields** (*Optional[Dict[str, Any]]*) – Fields which should be updated for the object.
- **async** (*Optional[bool]*) – True to add the request to the queue, so it can be executed later using async_run()
- **jira** (*jira.client.JIRA*) – Instance of Jira Client
- **notify** (*bool*) – True to notify watchers about the update, sets parameter notifyUsers. (Default: True). Admin or project admin permissions are required to disable the notification.
- **kwargs** (*Any*) – extra arguments to the PUT request.

delete(*params: dict[str, Any] | None = None*) → Response | None

Delete this resource from the server, passing the specified query parameters.

If this resource doesn't support DELETE, a *JIRAError* will be raised; subclasses that specialize this method will only raise errors in case of user error.

Parameters
params – Parameters for the delete request.

Returns
Optional[Response] – Returns None if async

_load(*url: str*, *headers={}*, *params: dict[str, str] | None = None*, *path: str | None = None*)

Load a resource.

Parameters

- **url** (*str*) – url
- **headers** (*Optional[CaseInsensitiveDict]*) – headers. Defaults to CaseInsensitiveDict().
- **params** (*Optional[Dict[str,str]]*) – params to get request. Defaults to None.
- **path** (*Optional[str]*) – field to get. Defaults to None.

Raises
ValueError – If json cannot be loaded

_parse_raw(*raw: dict[str, Any]*)

Parse a raw dictionary to create a resource.

Parameters
raw (*Dict[str, Any]*) –

_default_headers(*user_headers*)

class jira.resources.**Issue**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A Jira issue.

class _IssueFields

Bases: AnyLike

class _Comment

Bases: object

__init__() → None

class _Worklog

Bases: object

__init__() → None

__init__()

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

update(*fields: dict[str, Any] = None*, *update: dict[str, Any] = None*, *async_: bool = None*, *jira:* JIRA *= None*, *notify: bool = True*, ***fieldargs*)

Update this issue on the server.

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.

Jira projects may contain many issue types. Some issue screens have different requirements for fields in an issue. This information is available through the *JIRA.editmeta()* method. Further examples are available here: https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+Example+-+Edit+issues

Parameters

- **fields** (*Dict[str,Any]*) – a dict containing field names and the values to use
- **update** (*Dict[str,Any]*) – a dict containing update the operations to apply
- **async** (*Optional[bool]*) – True to add the request to the queue, so it can be executed later using async_run() (Default: None))
- **jira** (*Optional[jira.client.JIRA]*) – JIRA instance.

- **notify** (*bool*) – True to notify watchers about the update, sets parameter notifyUsers. (Default: `True`). Admin or project admin permissions are required to disable the notification.
- **fieldargs** (*dict*) – keyword arguments will generally be merged into fields, except lists, which will be merged into updates

get_field(*field_name: str*) → Any

Obtain the (parsed) value from the Issue's field.

Parameters
field_name (*str*) – The name of the field to get

Raises
AttributeError – If the field does not exist or if the field starts with an _

Returns
Any – Returns the parsed data stored in the field. For example, "project" would be of class *Project*

add_field_value(*field: str*, *value: str*)

Add a value to a field that supports multiple values, without resetting the existing values.

This should work with: labels, multiple checkbox lists, multiple select

Parameters

- **field** (*str*) – The field name
- **value** (*str*) – The field's value

delete(*deleteSubtasks=False*)

Delete this issue from the server.

Parameters
deleteSubtasks (*bool*) – True to also delete subtasks. If any are present the Issue won't be deleted (Default: `True`)

permalink()

Get the URL of the issue, the browsable one not the REST one.

Returns
str – URL of the issue

class jira.resources.**Comment**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

An issue comment.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

update(*fields: dict[str, Any] | None = None*, *async_: bool | None = None*, *jira:* JIRA *= None*, *body: str = ''*, *visibility: dict[str, str] | None = None*, *is_internal: bool = False*, *notify: bool = True*)

Update a comment.

Keyword arguments are marshalled into a dict before being sent.

Parameters

- **fields** (*Optional[Dict[str, Any]]*) – DEPRECATED => a comment doesn't have fields
- **async** (*Optional[bool]*) – True to add the request to the queue, so it can be executed later using async_run() (Default: `None`))
- **jira** (*jira.client.JIRA*) – Instance of Jira Client

- **visibility** (*Optional[Dict[str, str]]*) – a dict containing two entries: "type" and "value". "type" is 'role' (or 'group' if the Jira server has configured comment visibility for groups) "value" is the name of the role (or group) to which viewing of this comment will be restricted.
- **body** (*str*) – New text of the comment
- **is_internal** (*bool*) – True to mark the comment as 'Internal' in Jira Service Desk (Default: `False`)
- **notify** (*bool*) – True to notify watchers about the update, sets parameter notifyUsers. (Default: `True`). Admin or project admin permissions are required to disable the notification.

class jira.resources.**Project**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A Jira project.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**Attachment**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

An issue attachment.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

get()

Return the file content as a string.

iter_content(*chunk_size=1024*)

Return the file content as an iterable stream.

class jira.resources.**Component**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A project component.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

delete(*moveIssuesTo: str | None = None*)

Delete this component from the server.

Parameters

moveIssuesTo – the name of the component to which to move any issues this component is applied

class jira.resources.**Dashboard**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A Jira dashboard.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**DashboardItemProperty**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A jira dashboard item.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

update(*dashboard_id: str*, *item_id: str*, *value: dict[str, Any]*) → *DashboardItemProperty*

Update this resource on the server.

Keyword arguments are marshalled into a dict before being sent. If this resource doesn't support PUT, a *JIRAError* will be raised; subclasses that specialize this method will only raise errors in case of user error.

Parameters

- **dashboard_id** (*str*) – The id if the dashboard.
- **item_id** (*str*) – The id of the dashboard item (DashboardGadget) to target.
- **value** (*dict[str, Any]*) – The value of the targeted property key.

Returns

DashboardItemProperty

delete(*dashboard_id: str*, *item_id: str*) → Response

Delete dashboard item property.

Parameters

- **dashboard_id** (*str*) – The id of the dashboard.
- **item_id** (*str*) – The id of the dashboard item (DashboardGadget).

Returns

Response

class jira.resources.**DashboardItemPropertyKey**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A jira dashboard item property key.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**Filter**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

An issue navigator filter.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**DashboardGadget**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A jira dashboard gadget.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

update(*dashboard_id: str*, *color: str | None = None*, *position: dict[str, Any] | None = None*, *title: str | None = None*) → *DashboardGadget*

Update this resource on the server.

Keyword arguments are marshalled into a dict before being sent. If this resource doesn't support PUT, a *JIRAError* will be raised; subclasses that specialize this method will only raise errors in case of user error.

Parameters

- **dashboard_id** (*str*) – The id of the dashboard to add the gadget to *required*.

- **color** (*str*) – The color of the gadget, should be one of: blue, red, yellow, green, cyan, purple, gray, or white.
- **ignore_uri_and_module_key_validation** (*bool*) – Whether to ignore the validation of the module key and URI. For example, when a gadget is created that is part of an application that is not installed.
- **position** (*dict[str, int]*) – A dictionary containing position information like - *{"column": 0, "row", 1}*.
- **title** (*str*) – The title of the gadget.

Returns
DashboardGadget

delete(*dashboard_id: str*) → Response

Delete gadget from dashboard.

Parameters
dashboard_id (*str*) – The id of the dashboard.

Returns
Response

class jira.resources.**Votes**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Vote information on an issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**PermissionScheme**(*options*, *session*, *raw=None*)

Bases: *Resource*

Permissionscheme information on a project.

__init__(*options*, *session*, *raw=None*)

class jira.resources.**Watchers**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Watcher information on an issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

delete(*username*)

Remove the specified user from the watchers list.

class jira.resources.**Worklog**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Worklog on an issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

delete(*adjustEstimate: str | None = 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

class jira.resources.**IssueLink**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Link between two issues.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**IssueLinkType**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Type of link between two issues.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**IssueProperty**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Custom data against an issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

_find_by_url(*url: str*, *params: dict[str, str] | None = None*)

class jira.resources.**IssueSecurityLevelScheme**(*options*, *session*, *raw=None*)

Bases: *Resource*

IssueSecurityLevelScheme information on a project.

__init__(*options*, *session*, *raw=None*)

class jira.resources.**IssueType**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Type of issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**IssueTypeScheme**(*options*, *session*, *raw=None*)

Bases: *Resource*

An issue type scheme.

__init__(*options*, *session*, *raw=None*)

class jira.resources.**NotificationScheme**(*options*, *session*, *raw=None*)

Bases: *Resource*

NotificationScheme information on a project.

__init__(*options*, *session*, *raw=None*)

class jira.resources.**Priority**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

Priority that can be set on an issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**PriorityScheme**(*options*, *session*, *raw=None*)

Bases: *Resource*

PriorityScheme information on a project.

__init__(*options*, *session*, *raw=None*)

class jira.resources.**Version**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A version of a project.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

delete(*moveFixIssuesTo=None*, *moveAffectedIssuesTo=None*)

Delete this project version from the server.

If neither of the arguments are specified, the version is removed from all issues it is attached to.

Parameters

- **moveFixIssuesTo** – in issues for which this version is a fix version, add this version to the fix version list
- **moveAffectedIssuesTo** – in issues for which this version is an affected version, add this version to the affected version list

update(***kwargs*)

Update this project version from the server. It is prior used to archive versions.

Refer to Atlassian REST API documentation.

Example

```
>> version_id = "10543"
>> version = JIRA("https://atlassian.org").version(version_id)
>> print(version.name)
"some_version_name"
>> version.update(name="another_name")
>> print(version.name)
"another_name"
>> version.update(archived=True)
>> print(version.archived)
True
```

class jira.resources.**WorkflowScheme**(*options*, *session*, *raw=None*)

Bases: *Resource*

WorkflowScheme information on a project.

__init__(*options*, *session*, *raw=None*)

class jira.resources.**Role**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A role inside a project.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

update(*users: str | list | tuple = None*, *groups: str | list | tuple = None*)

Add the specified users or groups to this project role. One of `users` or `groups` must be specified.

Parameters

- **users** (*Optional[Union[str,List,Tuple]]*) – a user or users to add to the role
- **groups** (*Optional[Union[str,List,Tuple]]*) – a group or groups to add to the role

add_user(*users: str | list | tuple = None*, *groups: str | list | tuple = None*)

Add the specified users or groups to this project role. One of `users` or `groups` must be specified.

Parameters

- **users** (*Optional[Union[str,List,Tuple]]*) – a user or users to add to the role
- **groups** (*Optional[Union[str,List,Tuple]]*) – a group or groups to add to the role

class jira.resources.**Resolution**(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

Bases: *Resource*

A resolution for an issue.

__init__(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

class jira.resources.**SecurityLevel**(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

Bases: *Resource*

A security level for an issue or project.

__init__(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

class jira.resources.**Status**(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

Bases: *Resource*

Status for an issue.

__init__(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

class jira.resources.**User**(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*, ***, *_query_param: str = 'username'*)

Bases: *Resource*

A Jira user.

__init__(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*, ***, *_query_param: str = 'username'*)

class jira.resources.**Group**(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

Bases: *Resource*

A Jira user group.

__init__(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

class jira.resources.**CustomFieldOption**(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

Bases: *Resource*

An existing option for a custom issue field.

__init__(*options: dict[str, str]*, *session: ResilientSession*, *raw: dict[str, Any] = None*)

class jira.resources.**RemoteLink**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A link to a remote application from an issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

update(*object*, *globalId=None*, *application=None*, *relationship=None*)

Update a RemoteLink. 'object' is required.

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

- **object** – 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)

class jira.resources.**Customer**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A Service Desk customer.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**ServiceDesk**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A Service Desk.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**RequestType**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A Service Desk Request Type.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**StatusCategory**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

StatusCategory for an issue.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**AgileResource**(*path: str*, *options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *Resource*

A generic Agile resource. Also known as Jira Agile Server, Jira Software and formerly GreenHopper.

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

AGILE_BASE_REST_PATH = 'agile'

Public API introduced in Jira Agile 6.7.7.

__init__(*path: str*, *options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**Sprint**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *AgileResource*

An Agile sprint.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

class jira.resources.**Board**(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

Bases: *AgileResource*

An Agile board.

__init__(*options: dict[str, str]*, *session:* ResilientSession, *raw: dict[str, Any] = None*)

6.1.7 jira.utils module

Jira utils used internally.

class jira.utils.**CaseInsensitiveDict**(**args*, ***kwargs*)

Bases: `CaseInsensitiveDict`

A case-insensitive `dict`-like object.

DEPRECATED: use requests.structures.CaseInsensitiveDict directly.

Implements all methods and operations of `collections.MutableMapping` as well as dict's `copy`. Also provides `lower_items`.

All keys are expected to be strings. The structure remembers the case of the last key to be set, and `iter(instance)`, `keys()`, `items()`, `iterkeys()` will contain case-sensitive keys. However, querying and contains testing is case insensitive:

```
cid = CaseInsensitiveDict()
cid['Accept'] = 'application/json'
cid['accept'] == 'application/json'  # True
list(cid) == ['Accept']  # True
```

For example, `headers['content-encoding']` will return the value of a `'Content-Encoding'` response header, regardless of how the header name was originally stored.

If the constructor, `.update`, or equality comparison operations are given keys that have equal `.lower()` s, the behavior is undefined.

__init__(**args*, ***kwargs*) → None

jira.utils.**threaded_requests**(*requests*)

jira.utils.**json_loads**(*resp: Response | None*) → Any

Attempts to load json the result of a response.

Parameters
resp (*Optional[Response]*) – The Response object

Raises
JIRAError – via *jira.resilientsession.raise_on_error()*

Returns
Union[List[Dict[str, Any]], Dict[str, Any]] – the json

jira.utils.**remove_empty_attributes**(*data: dict[str, Any]*) → dict[str, Any]

A convenience function to remove key/value pairs with *None* for a value.

Parameters
data – A dictionary.

Returns
Dict[str, Any] – A dictionary with no *None* key/value pairs.

This documents the `jira` python package (version 3.8.1.dev5+geb0ec90), a Python library designed to ease the use of the Jira REST API. Some basic support for the Jira Agile / Jira Software REST API also exists.

Documentation is also available in Dash format.

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

The release history and notes and can be found at https://github.com/pycontribs/jira/releases

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

j

jira.client, 25
jira.config, 61
jira.exceptions, 62
jira.jirashell, 63
jira.resilientsession, 63
jira.resources, 65
jira.utils, 76

INDEX

Symbols

_HASH_IDS (*jira.resources.Resource attribute*), 65
_READABLE_IDS (*jira.resources.Resource attribute*), 65
__init__() (*jira.client.JIRA method*), 27
__init__() (*jira.client.JiraCookieAuth method*), 26
__init__() (*jira.client.QshGenerator method*), 26
__init__() (*jira.client.ResultList method*), 26
__init__() (*jira.client.TokenAuth method*), 27
__init__() (*jira.exceptions.JIRAError method*), 62
__init__() (*jira.exceptions.NotJIRAInstanceError method*), 62
__init__() (*jira.resilientsession.ResilientSession method*), 64
__init__() (*jira.resources.AgileResource method*), 76
__init__() (*jira.resources.Attachment method*), 69
__init__() (*jira.resources.Board method*), 76
__init__() (*jira.resources.Comment method*), 68
__init__() (*jira.resources.Component method*), 69
__init__() (*jira.resources.CustomFieldOption method*), 74
__init__() (*jira.resources.Customer method*), 75
__init__() (*jira.resources.Dashboard method*), 69
__init__() (*jira.resources.DashboardGadget method*), 70
__init__() (*jira.resources.DashboardItemProperty method*), 69
__init__() (*jira.resources.DashboardItemPropertyKey method*), 70
__init__() (*jira.resources.Filter method*), 70
__init__() (*jira.resources.Group method*), 74
__init__() (*jira.resources.Issue method*), 67
__init__() (*jira.resources.Issue._IssueFields method*), 67
__init__() (*jira.resources.Issue._IssueFields._Comment method*), 67
__init__() (*jira.resources.Issue._IssueFields._Worklog method*), 67
__init__() (*jira.resources.IssueLink method*), 72
__init__() (*jira.resources.IssueLinkType method*), 72
__init__() (*jira.resources.IssueProperty method*), 72
__init__() (*jira.resources.IssueSecurityLevelScheme method*), 72
__init__() (*jira.resources.IssueType method*), 72
__init__() (*jira.resources.IssueTypeScheme method*), 72
__init__() (*jira.resources.NotificationScheme method*), 72
__init__() (*jira.resources.PermissionScheme method*), 71
__init__() (*jira.resources.Priority method*), 72
__init__() (*jira.resources.PriorityScheme method*), 73
__init__() (*jira.resources.Project method*), 69
__init__() (*jira.resources.RemoteLink method*), 75
__init__() (*jira.resources.RequestType method*), 75
__init__() (*jira.resources.Resolution method*), 74
__init__() (*jira.resources.Resource method*), 65
__init__() (*jira.resources.Role method*), 73
__init__() (*jira.resources.SecurityLevel method*), 74
__init__() (*jira.resources.ServiceDesk method*), 75
__init__() (*jira.resources.Sprint method*), 76
__init__() (*jira.resources.Status method*), 74
__init__() (*jira.resources.StatusCategory method*), 75
__init__() (*jira.resources.User method*), 74
__init__() (*jira.resources.Version method*), 73
__init__() (*jira.resources.Votes method*), 71
__init__() (*jira.resources.Watchers method*), 71
__init__() (*jira.resources.WorkflowScheme method*), 73
__init__() (*jira.resources.Worklog method*), 71
__init__() (*jira.utils.CaseInsensitiveDict method*), 76
_default_headers() (*jira.resources.Resource method*), 67
_find_by_url() (*jira.resources.IssueProperty method*), 72
_find_by_url() (*jira.resources.Resource method*), 66
_get_url() (*jira.resources.Resource method*), 66
_load() (*jira.resources.Resource method*), 66
_parse_raw() (*jira.resources.Resource method*), 67

A

add_attachment() (*jira.client.JIRA method*), 30
add_comment() (*jira.client.JIRA method*), 38
add_field_value() (*jira.resources.Issue method*), 68
add_gadget_to_dashboard() (*jira.client.JIRA method*), 33
add_group() (*jira.client.JIRA method*), 34
add_issue_property() (*jira.client.JIRA method*), 43

add_issues_to_epic() (*jira.client.JIRA method*), 60
add_issues_to_sprint() (*jira.client.JIRA method*), 60
add_remote_link() (*jira.client.JIRA method*), 39
add_simple_link() (*jira.client.JIRA method*), 39
add_user() (*jira.client.JIRA method*), 57
add_user() (*jira.resources.Role method*), 74
add_user_to_group() (*jira.client.JIRA method*), 57
add_vote() (*jira.client.JIRA method*), 41
add_watcher() (*jira.client.JIRA method*), 41
add_worklog() (*jira.client.JIRA method*), 42
AGILE_BASE_REST_PATH (*jira.resources.AgileResource attribute*), 75
AGILE_BASE_URL (*jira.client.JIRA attribute*), 27
AGILE_BASE_URL (*jira.resources.AgileResource attribute*), 75
AgileResource (*class in jira.resources*), 75
all_dashboard_gadgets() (*jira.client.JIRA method*), 33
application_properties() (*jira.client.JIRA method*), 30
applicationlinks() (*jira.client.JIRA method*), 30
assign_issue() (*jira.client.JIRA method*), 37
async_do() (*jira.client.JIRA method*), 30
Attachment (*class in jira.resources*), 69
attachment() (*jira.client.JIRA method*), 30
attachment_meta() (*jira.client.JIRA method*), 30
avatars() (*jira.client.JIRA method*), 56

B

backup() (*jira.client.JIRA method*), 55
backup_complete() (*jira.client.JIRA method*), 55
backup_download() (*jira.client.JIRA method*), 55
backup_progress() (*jira.client.JIRA method*), 55
Board (*class in jira.resources*), 76
boards() (*jira.client.JIRA method*), 58

C

CaseInsensitiveDict (*class in jira.utils*), 76
checked_version (*jira.client.JIRA attribute*), 27
client_info() (*jira.client.JIRA method*), 29
close() (*jira.client.JIRA method*), 29
cloud_api() (*in module jira.client*), 25
Comment (*class in jira.resources*), 68
comment() (*jira.client.JIRA method*), 38
comments() (*jira.client.JIRA method*), 37
Component (*class in jira.resources*), 69
component() (*jira.client.JIRA method*), 31
component_count_related_issues() (*jira.client.JIRA method*), 31
confirm_project_avatar() (*jira.client.JIRA method*), 47
confirm_user_avatar() (*jira.client.JIRA method*), 51
cookies (*jira.client.JiraCookieAuth property*), 26
copy_dashboard() (*jira.client.JIRA method*), 32
create_board() (*jira.client.JIRA method*), 59
create_component() (*jira.client.JIRA method*), 31
create_customer() (*jira.client.JIRA method*), 35
create_customer_request() (*jira.client.JIRA method*), 36
create_dashboard() (*jira.client.JIRA method*), 32
create_filter() (*jira.client.JIRA method*), 33
create_issue() (*jira.client.JIRA method*), 35
create_issue_link() (*jira.client.JIRA method*), 43
create_issues() (*jira.client.JIRA method*), 35
create_project() (*jira.client.JIRA method*), 56
create_sprint() (*jira.client.JIRA method*), 60
create_temp_project_avatar() (*jira.client.JIRA method*), 46
create_temp_user_avatar() (*jira.client.JIRA method*), 51
create_version() (*jira.client.JIRA method*), 53
createmeta() (*jira.client.JIRA method*), 37
createmeta_fieldtypes() (*jira.client.JIRA method*), 36
createmeta_issuetypes() (*jira.client.JIRA method*), 36
current_user() (*jira.client.JIRA method*), 55
custom_field_option() (*jira.client.JIRA method*), 31
Customer (*class in jira.resources*), 75
CustomFieldOption (*class in jira.resources*), 74

D

Dashboard (*class in jira.resources*), 69
dashboard() (*jira.client.JIRA method*), 32
dashboard_gadgets() (*jira.client.JIRA method*), 33
dashboard_item_property() (*jira.client.JIRA method*), 32
dashboard_item_property_keys() (*jira.client.JIRA method*), 32
DashboardGadget (*class in jira.resources*), 70
DashboardItemProperty (*class in jira.resources*), 69
DashboardItemPropertyKey (*class in jira.resources*), 70
dashboards() (*jira.client.JIRA method*), 32
deactivate_user() (*jira.client.JIRA method*), 54
DEFAULT_OPTIONS (*jira.client.JIRA attribute*), 27
delete() (*jira.resources.Component method*), 69
delete() (*jira.resources.DashboardGadget method*), 71
delete() (*jira.resources.DashboardItemProperty method*), 70
delete() (*jira.resources.Issue method*), 68
delete() (*jira.resources.Resource method*), 66
delete() (*jira.resources.Version method*), 73
delete() (*jira.resources.Watchers method*), 71
delete() (*jira.resources.Worklog method*), 71
delete_attachment() (*jira.client.JIRA method*), 31
delete_board() (*jira.client.JIRA method*), 59
delete_component() (*jira.client.JIRA method*), 31
delete_issue_link() (*jira.client.JIRA method*), 44
delete_permissionscheme() (*jira.client.JIRA method*), 56

delete_project() (*jira.client.JIRA method*), 55
delete_project_avatar() (*jira.client.JIRA method*), 47
delete_remote_link() (*jira.client.JIRA method*), 52
delete_screen() (*jira.client.JIRA method*), 56
delete_user() (*jira.client.JIRA method*), 54
delete_user_avatar() (*jira.client.JIRA method*), 52

E

editmeta() (*jira.client.JIRA method*), 38
experimental_atlassian_api() (*in module jira.client*), 25

F

favourite_filters() (*jira.client.JIRA method*), 33
fields() (*jira.client.JIRA method*), 33
Filter (*class in jira.resources*), 70
filter() (*jira.client.JIRA method*), 33
find() (*jira.client.JIRA method*), 29
find() (*jira.resources.Resource method*), 65
find_transitionid_by_name() (*jira.client.JIRA method*), 40

G

get() (*jira.resources.Attachment method*), 69
get_config() (*in module jira.jirashell*), 63
get_field() (*jira.resources.Issue method*), 68
get_igrid() (*jira.client.JIRA method*), 58
get_issue_type_scheme_associations() (*jira.client.JIRA method*), 56
get_jira() (*in module jira.config*), 61
get_project_version_by_name() (*jira.client.JIRA method*), 47
Group (*class in jira.resources*), 74
group() (*jira.client.JIRA method*), 34
group_members() (*jira.client.JIRA method*), 34
groups() (*jira.client.JIRA method*), 34

H

handle_401() (*jira.client.JiraCookieAuth method*), 26
handle_basic_auth() (*in module jira.jirashell*), 63

I

incompletedIssuesEstimateSum() (*jira.client.JIRA method*), 59
init_session() (*jira.client.JiraCookieAuth method*), 26
Issue (*class in jira.resources*), 67
issue() (*jira.client.JIRA method*), 34
Issue._IssueFields (*class in jira.resources*), 67
Issue._IssueFields._Comment (*class in jira.resources*), 67
Issue._IssueFields._Worklog (*class in jira.resources*), 67
issue_link() (*jira.client.JIRA method*), 44
issue_link_type() (*jira.client.JIRA method*), 44
issue_link_types() (*jira.client.JIRA method*), 44
issue_properties() (*jira.client.JIRA method*), 43
issue_property() (*jira.client.JIRA method*), 43
issue_type() (*jira.client.JIRA method*), 45
issue_type_by_name() (*jira.client.JIRA method*), 45
issue_type_schemes() (*jira.client.JIRA method*), 56
issue_types() (*jira.client.JIRA method*), 44
issue_types_for_project() (*jira.client.JIRA method*), 49
IssueLink (*class in jira.resources*), 72
IssueLinkType (*class in jira.resources*), 72
IssueProperty (*class in jira.resources*), 72
IssueSecurityLevelScheme (*class in jira.resources*), 72
issuesecurityschemes() (*jira.client.JIRA method*), 56
IssueType (*class in jira.resources*), 72
IssueTypeScheme (*class in jira.resources*), 72
iter_content() (*jira.resources.Attachment method*), 69

J

JIRA (*class in jira.client*), 27
jira.client
    module, 25
jira.config
    module, 61
jira.exceptions
    module, 62
jira.jirashell
    module, 63
jira.resilientsession
    module, 63
jira.resources
    module, 65
jira.utils
    module, 76
JIRA_BASE_URL (*jira.client.JIRA attribute*), 27
JIRA_BASE_URL (*jira.resources.Resource attribute*), 65
JiraCookieAuth (*class in jira.client*), 26
JIRAError, 62
json_loads() (*in module jira.utils*), 76

K

kill_session() (*jira.client.JIRA method*), 54
kill_websudo() (*jira.client.JIRA method*), 54

M

main() (*in module jira.jirashell*), 63
module
    jira.client, 25
    jira.config, 61
    jira.exceptions, 62
    jira.jirashell, 63
    jira.resilientsession, 63
    jira.resources, 65
    jira.utils, 76
move_to_backlog() (*jira.client.JIRA method*), 61
move_version() (*jira.client.JIRA method*), 53

my_permissions() (*jira.client.JIRA method*), 45
myself() (*jira.client.JIRA method*), 49

N

NotificationScheme (*class in jira.resources*), 72
notificationschemes() (*jira.client.JIRA method*), 56
NotJIRAInstanceError, 62

O

oauth_dance() (*in module jira.jirashell*), 63

P

parse_error_msg() (*in module jira.resilientsession*), 64
parse_errors() (*in module jira.resilientsession*), 63
PassthroughRetryPrepare (*class in jira.resilientsession*), 63
permalink() (*jira.resources.Issue method*), 68
PermissionScheme (*class in jira.resources*), 71
permissionschemes() (*jira.client.JIRA method*), 56
prepare() (*jira.resilientsession.PassthroughRetryPrepare method*), 63
prepare() (*jira.resilientsession.PrepareRequestForRetry method*), 63
PrepareRequestForRetry (*class in jira.resilientsession*), 63
priorities() (*jira.client.JIRA method*), 46
Priority (*class in jira.resources*), 72
priority() (*jira.client.JIRA method*), 46
PriorityScheme (*class in jira.resources*), 72
process_command_line() (*in module jira.jirashell*), 63
process_config() (*in module jira.jirashell*), 63
process_original_request() (*jira.client.JiraCookieAuth method*), 26
Project (*class in jira.resources*), 69
project() (*jira.client.JIRA method*), 46
project_avatars() (*jira.client.JIRA method*), 46
project_components() (*jira.client.JIRA method*), 47
project_issue_fields() (*jira.client.JIRA method*), 44
project_issue_security_level_scheme() (*jira.client.JIRA method*), 40
project_issue_types() (*jira.client.JIRA method*), 44
project_notification_scheme() (*jira.client.JIRA method*), 40
project_permissionscheme() (*jira.client.JIRA method*), 41
project_priority_scheme() (*jira.client.JIRA method*), 41
project_role() (*jira.client.JIRA method*), 48
project_roles() (*jira.client.JIRA method*), 48
project_versions() (*jira.client.JIRA method*), 47
project_workflow_scheme() (*jira.client.JIRA method*), 41
projectcategories() (*jira.client.JIRA method*), 56
projects() (*jira.client.JIRA method*), 46

Q

QshGenerator (*class in jira.client*), 26

R

raise_on_error() (*in module jira.resilientsession*), 63
rank() (*jira.client.JIRA method*), 61
reindex() (*jira.client.JIRA method*), 54
remote_link() (*jira.client.JIRA method*), 38
remote_links() (*jira.client.JIRA method*), 38
RemoteLink (*class in jira.resources*), 74
remove_empty_attributes() (*in module jira.utils*), 76
remove_group() (*jira.client.JIRA method*), 34
remove_user_from_group() (*jira.client.JIRA method*), 57
remove_vote() (*jira.client.JIRA method*), 41
remove_watcher() (*jira.client.JIRA method*), 42
removed_issues() (*jira.client.JIRA method*), 59
removedIssuesEstimateSum() (*jira.client.JIRA method*), 59
rename_user() (*jira.client.JIRA method*), 54
rename_version() (*jira.client.JIRA method*), 48
request() (*jira.resilientsession.ResilientSession method*), 64
request_type_by_name() (*jira.client.JIRA method*), 45
request_types() (*jira.client.JIRA method*), 45
RequestType (*class in jira.resources*), 75
ResilientSession (*class in jira.resilientsession*), 64
Resolution (*class in jira.resources*), 74
resolution() (*jira.client.JIRA method*), 48
resolutions() (*jira.client.JIRA method*), 48
Resource (*class in jira.resources*), 65
ResourceType (*in module jira.client*), 64
ResultList (*class in jira.client*), 25
Role (*class in jira.resources*), 73
role() (*jira.client.JIRA method*), 58

S

screens() (*jira.client.JIRA method*), 56
search_allowed_users_for_issue() (*jira.client.JIRA method*), 53
search_assignable_users_for_issues() (*jira.client.JIRA method*), 50
search_assignable_users_for_projects() (*jira.client.JIRA method*), 50
search_issues() (*jira.client.JIRA method*), 48
search_users() (*jira.client.JIRA method*), 52
security_level() (*jira.client.JIRA method*), 49
SecurityLevel (*class in jira.resources*), 74
send_request() (*jira.client.JiraCookieAuth method*), 26
server_info() (*jira.client.JIRA method*), 49
server_url (*jira.client.JIRA property*), 29
service_desk() (*jira.client.JIRA method*), 36

service_desks() (*jira.client.JIRA method*), 35
ServiceDesk (*class in jira.resources*), 75
session() (*jira.client.JIRA method*), 54
set_application_property() (*jira.client.JIRA method*), 30
set_dashboard_item_property() (*jira.client.JIRA method*), 32
set_project_avatar() (*jira.client.JIRA method*), 47
set_user_avatar() (*jira.client.JIRA method*), 52
Sprint (*class in jira.resources*), 76
sprint() (*jira.client.JIRA method*), 59
sprint_info() (*jira.client.JIRA method*), 59
sprints() (*jira.client.JIRA method*), 58
sprints_by_name() (*jira.client.JIRA method*), 58
Status (*class in jira.resources*), 74
status() (*jira.client.JIRA method*), 49
statuscategories() (*jira.client.JIRA method*), 49
StatusCategory (*class in jira.resources*), 75
statuscategory() (*jira.client.JIRA method*), 50
statuses() (*jira.client.JIRA method*), 49
supports_service_desk() (*jira.client.JIRA method*), 35

T

templates() (*jira.client.JIRA method*), 56
threaded_requests() (*in module jira.utils*), 76
TokenAuth (*class in jira.client*), 26
transition_issue() (*jira.client.JIRA method*), 40
transitions() (*jira.client.JIRA method*), 39
translate_resource_args() (*in module jira.client*), 25

U

update() (*jira.resources.Comment method*), 68
update() (*jira.resources.DashboardGadget method*), 70
update() (*jira.resources.DashboardItemProperty method*), 70
update() (*jira.resources.Issue method*), 67
update() (*jira.resources.RemoteLink method*), 75
update() (*jira.resources.Resource method*), 66
update() (*jira.resources.Role method*), 73
update() (*jira.resources.Version method*), 73
update_cookies() (*jira.client.JiraCookieAuth method*), 26
update_dashboard_automatic_refresh_minutes() (*jira.client.JIRA method*), 32
update_filter() (*jira.client.JIRA method*), 33
update_sprint() (*jira.client.JIRA method*), 59
User (*class in jira.resources*), 74
user() (*jira.client.JIRA method*), 50
user_avatars() (*jira.client.JIRA method*), 51

V

Version (*class in jira.resources*), 73
version() (*jira.client.JIRA method*), 53
version_count_related_issues() (*jira.client.JIRA method*), 54
version_count_unresolved_issues() (*jira.client.JIRA method*), 54
Votes (*class in jira.resources*), 71
votes() (*jira.client.JIRA method*), 40

W

Watchers (*class in jira.resources*), 71
watchers() (*jira.client.JIRA method*), 41
workflows() (*jira.client.JIRA method*), 56
WorkflowScheme (*class in jira.resources*), 73
workflowscheme() (*jira.client.JIRA method*), 56
Worklog (*class in jira.resources*), 71
worklog() (*jira.client.JIRA method*), 42
worklogs() (*jira.client.JIRA method*), 42