ZENDESK API WRAPPER DOCUMENTATION @ BroadConnect

***NOTE:** many models have standard/common methods not documentated here, such as “put”, “delete”, and “get”. Please refer to “endpoints.py” to see which methods are supported on a model

****NOTE:** fetch and query return a Helper object, see HELPER section below

VIEW:

```
view = View.get(api, view_id=30583090)
views = View.fetch(api).all()
view_tickets = view.get_tickets(api).all()
```

VIEW_COUNT:

```
view_counts = View_Count.fetch(api, ids=[123123,123123,123123])
```

TICKET:

```
audits = ticket.get_audits(api).all()
comments = ticket.get_comments(api).all()
tickets = Ticket.query(api).all()
ticket = Ticket.get(api, ticket_id=10241)

ticket = Ticket(name='test', description='test')
t = ticket.put(api)
ticket.name='test'#UPDATE EXAMPLE
t.put(api)

ticket.delete(api)
t.delete(api)
```

TICKET_METRIC:

```
metrics = Ticket_Metric.fetch(api).all()
metric = Ticket_Metric.get(api, ticket_metric_id=466323574)
```

COMMENT:

```
comments = Comment.fetch(api, ticket_id=5180).all()
```

AUDIT:

```
audits = Audit.fetch(api, ticket_id=5180).all()
audit = Audit.get(api, audit_id=11222380740, ticket_id=5180)
```

TICKET_FORM:

```
ticket_forms = Ticket_Form.fetch(api).all()
ticket_form = Ticket_Form.get(api, ticket_form_id=11924)
ticket_fields = ticket_form.get_fields(api)
```

TICKET_FIELD:

```
ticket_fields = Ticket_Field.fetch(api).all()
```

USERS:

```
users = User.fetch(api, ids=[304251889, 304251889]).all()
users = Users.query(api, query=[('name','like','evan')]).all()
users = Users.query(api).all()
user = User.get(api, user_id=304251889)
user_memberships = user.get_memberships(api).all()
```

USER_FIELD:

```
user_field = User_Field.fetch(api).all()
```

GROUP:

```
group = Group.get(api, group_id=20708879)
groups = Group.query(api).all()

group_memberships = group.get_memberships(api).all()
```

GROUP_MEMBERSHIP:

```
group_memberships = Group_Membership.fetch(api, group_id=13123)

user_memberships = Group_Membership.fetch(api, user_id=13123)

memberships = Group_Membership.fetch(api)
membership = Group_Membership.get(api, group_membership_id=21212829)
```

CUSTOM_ROLE:

```
custom_roles = Custom_Role.fetch(api).all()
```

IDENTITIE:

```
user_identities = Identitie.fetch(api, user_id=12312312).all()
identity = Identitie.get(api, user_id=123, identity_id=123)

identity = Identitie(type='email', value='someone@example.com')
identity.put(api, user_id=304251889)

identity = Identitie.get(api, user_id=304251949, identity_id=386981060)
identity.delete(api, user_id=304251949)
```

ORGANIZATION:

```
organizations = Organization.fetch(api).all()
organization = Organization.get(api, organization_id=23882965)
organization = Organization(name="name")
organization.put(api)
organization.name = 'update name'
organization.put(api)
```

ORGANIZATION_FIELD:

```
organization_fields = Organization_Field.fetch(api).all()
organization_field = Organization_Field.get(api, organization_field_id=18009)
organization_field = Organization_Field(type='text',title='test',
                                        description='test', position=0,
                                        active=True, key='test')
organization_field.put(api)
```

HELPER:

Helper works on collections of objects and works on FETCH and QUERY method calls

ie:

```
query = Ticket.query(api)
for item in query.all():
    print item.id

if query.has_next():
    next_query = query.next(api)

    for item in next_query.all():
        print item.id
```

ie:

```
tickets = None
while True:
    if tickets is None:
        tickets = Ticket.query(api) //or Ticket.fetch(api)

    for t in tickets.all():
        print t.id
    if not tickets.has_next():
        break
    tickets = tickets.next(api)
```

supported methods:

```
fetch.all() //gets all items
fetch.get_sideloaded() //gets any sideloaded items
fetch.has_next() //if there is a next page
fetch.has_prev() //if there is a previous page
fetch.next() //makes call for next page (after, call all() to get results)
fetch.prev() //makes call for previous page (after, call all() to get results)
fetch.count() //gets the amount of items returned in the request

fetch.next_page //the next page as a URL
fetch.prev_page //the prev page as a URL
```

A PAGINATION EXAMPLE:

tickets = Ticket.query(api, url='https://broadconnectusa.zendesk.com/api/v2/search.json?page=2&per_page=2&query=type%3Aticket')

UPLOAD:

- to upload one or more attachments, supply a multi key value in the following format:

files =[{'file':open('pdf-test.pdf', 'rb'), 'name':'pdf-test.pdf', 'mimetype':'application/pdf'}]

tokens, attachments = Upload.upload(api, files=files)

ATTACHMENT:

attachment = Attachment.get(api, attachment_id=350688274)

ADDING ATTACHMENT TO A TICKET:

1) upload the files

PDF:
files =[{'file':open('pdf-test.pdf', 'rb'), 'name':'pdf-test.pdf', 'mimetype':'application/pdf'}]

or

PNG image file:
files =[{'file':open('test2.png', 'rb'), 'name':'test2.png', 'mimetype':'image/png'}]

*NOTE: specify mimetype

tokens, attachments = Upload.upload(api, files=files)

2) pass list of tokens to the comment

```
ticket = Ticket(name='this ticket will include attachments', comment=
                                    { 'value': 'this will have the attachments', 'uploads':  tokens }
                )
```

ticket.put(api)

QUERY (method):

- a global method that works with a few models such as TICKET, USER, ORGANIZATION etc...
- utilizes the zendesk SEARCH REST API call
- by default returns all results
- query argument supports a more complex search query
- query takes in a list of tuples in the following format: ('name','like', 'test')
- current operations supported include “=”, “<”, “>”, “like”

ie:

tickets = Ticket.query(api, query=[('name','like', 'test')]).all()

EXCEPTIONS AND SPECIAL CASE SCENARIOS:

1) if first-level results “key” returned do not match the model name, such as from a GET, in order to instantiate an object based on the results, kwargs should be updated with an “override” parameter to that first-level “key”. This is not very common but because Zendesk is not consistent with names, this can happen in special cases. See “IDENTITIE” model for an example of how to override the GET method to handle this case.