

---

# Catalist Documentation

*Release 0.0*

**Rachel Wu, Tony Zhang**

January 22, 2016



## CONTENTS

<b>1</b>	<b>Indices and tables</b>	<b>5</b>
	<b>Python Module Index</b>	<b>7</b>
	<b>Index</b>	<b>9</b>



Contents:

**class** routes.**Catalist** (\*args, \*\*values)  
A class for our lists (Catalists :P)

**class** routes.**CatalistEntry** (\*args, \*\*kwargs)  
A class for the entries in our Catalists

**class** routes.**CatalistKVP** (\*args, \*\*kwargs)  
A class for individual key-value pairs in our Catalist entries

**exception** routes.**InvalidAPIUsage** (message, status\_code=None, payload=None)  
A class for exceptions to raise in invalid API usage. Shamelessly pillaged from [Flask's documentation](#)

**class** routes.**Role** (\*args, \*\*values)  
A class for user roles (e.g. user, admin, ...)

**class** routes.**User** (\*args, \*\*values)  
A class for users. Can have any/none of these attributes.

routes.**about** ()  
About us.

routes.**add\_to\_my\_lists** ()  
Add a specified list to "My Lists". POST {  
    listid: <listid>  
}

routes.**admin\_unames** = ['rmwu', 'txz']  
Currently admins are determined by residency on this list. Hacky, I know. \_\_\_\_.

routes.**autocomplete** ()  
completes a word fragment with a possible list type usage: POST to this route with {"fragment": myfragment}, response is the list of possible completions of *myfragment* drawn from *autocomplete\_dict*

routes.**cmp\_permission** (perm1, perm2)  
Return a positive/0/negative integer when perm1 >=/< perm2

routes.**create\_list** ()  
Create a new list and return the assigned listid  
Returns: the assigned listid

routes.**entry\_delete** ()  
Delete an entry from a Catalist. Requires at least edit permission.  
usage: POST a JSON associative array as follows: {  
    listid: <the id of the Catalist>, entryind: <the index of the entry to remove>  
}

routes.**entry\_title\_save** ()  
AJAXily save the title of an entry. Requires at least edit permission  
usage: POST a JS associative array (basically a dict) like so: {  
    listid: <the list id>, entryind: <index of entry w.r.t. list (0-indexing)>, newvalue: <new entry title>  
}

routes.**get\_id** ()  
Return name of current user

`routes.get_pref (*args, **kwargs)`  
Get the preferred theme for the user POST: {  
    uid: <uid>,  
} returns: {  
    theme: <preferred theme [0, 1, ...]>  
}

`routes.getlist (listid)`  
Display a list with given listid from our database.

`routes.human_readable_time_since (tiem)`  
Give a human-readable representation of time elapsed since a given time  
**Parameters** `tiem` – a `datetime` object representing the given time.

`routes.index ()`  
Our homepage!

`routes.isLoggedIn ()`  
Used for .js to call

`routes.key_save ()`  
Save a key. Requires at least edit permission.  
POST a JS associative array (basically a dict) like so: {  
    listid: <the list id>, entryind: <index of entry w.r.t. list (0-indexing)>, index: <index of key-val pair w.r.t. entry>, newvalue: <new value of key>  
}

`routes.key_val_save (req_form, key_or_val)`  
Save a key or value in a KVP. Auxillary method for `/api/savekey` and `/api/savevalue` – captures repetitive code.

`routes.kvp_delete ()`  
Delete a key-value pair from a Catalist entry. Requires at least edit permission.  
usage: POST a JSON associative array as follows: {  
    listid: <the id of the Catalist>, entryind: <the index of the entry to remove>, index: <the index of the kvp within the entry>  
}

`routes.list_delete ()`  
Delete a Catalist. Requires at least own permission  
usage: POST a JSON associative array as follows: {  
    listid: <the id of the list to be deleted>  
}

`routes.list_save ()`  
Save an entire list. If listid is provided, the list is written onto the referenced list. Otherwise, a new list is created. In both cases the listid to which we saved the list is returned.  
usage: {  
    title: <thetitle>, contents: [  
        [title, [ attrname, attrval], ... ]

] (optionally) , listid: <the listid to save to>  
}

Returns: the given or assigned listid

`routes.list_title_save()`

AJAXily save the title of a Catalist ^\_^ Requires at least edit permission.

usage: POST a JS assoc array like so: {

listid: <the list id>, newvalue: <our new title>

}

`routes.login()`

Page for user login

`routes.logout()`

Log out the current user, clearing the Remember Me cookie

`routes.make_list()`

Upon making the first edit, an empty list will be created for the insertion of more data

`routes.my_lists_interact(listid, addQ)`

Add or remove a list with specified listid from “My Lists”.

#### Parameters

- **listid** – the listid of the list
- **addQ** – an integer specifying whether to add or remove: 1 to add, -1 to remove

`routes.perm_list = ['none', 'view', 'edit', 'own', 'admin']`

A list of all permission levels, from lowest to highest. The levels:

- 1.none – no permission
- 2.view – permission to view a list
- 3.edit – permission to edit a list
- 4.own – permission to change permission for a list
- 5.admin – can do anything

`routes.permissions_get()`

Get the permission level a user has for a particular list.

usage: POST the following: {

listid: <the listid>

}

returns: {

permission: <the current permission>

}

`routes.permissions_set()`

{ Set permissions for a user. Requires at least own permission

listid: <listid>, target: <username of user to set perms with>, permission: { none | view | edit | own | admin }

}

```
routes.preview_list (listid)
    Fetch the list with given listid from our database, display with template

routes.public_level_set ()
    Set the permission level for a list for the public at-large. Requires own permission.

    POST: { listid: <the listid>, permission: { none | view | edit | own | admin }
    }

routes.query_cur_perm (catalist)
    Finds the permission the current user has for list catalist

routes.query_permission (user, catalist)
    Gives the permission level a user has for a list. "None" represents an anonymous user.

routes.register ()
    Page for user registration

routes.remove_from_my_lists ()
    Remove a specified list from "My Lists". POST {
        listid: <listid>
    }

routes.signin ()
    Sign the user in, given valid credentials.

routes.signup ()
    Sign the user up, given valid credentials and a username the doesn't already exist in our database.

routes.userlists (*args, **kwargs)
    A page displaying all lists belonging to the user.

routes.value_save ()
    Save the value in a particular key-value pair. Requires at least edit permission.
    The API is virtually identical the that of key_save()

routes.vote ()
    Two options: 1. Update the database to incorporate a user's vote on an entry. 2. Find the user's current vote and
    the current score of the entry. Requires at least view permission
    usage: POST the following {
        listid: <listid>, entryind: <entryind>, userid: <userid>, vote: { 1 (upvote) | 0 (no vote) |
        -1 (downvote) | 100 (get the current vote) }
    }

    Returns a response with the following forms:

    if vote == 100 { current_vote: <the user's current vote>, score: <the entry's current score>
    } if vote != 100 {
        current_vote: <the vote just made>, score: <the entry's new score>
    }
```



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



**r**

routes, [1](#)



**A**

about() (in module routes), 1  
 add\_to\_my\_lists() (in module routes), 1  
 admin\_unames (in module routes), 1  
 autocomplete() (in module routes), 1

**C**

Catalist (class in routes), 1  
 CatalistEntry (class in routes), 1  
 CatalistKVP (class in routes), 1  
 cmp\_permission() (in module routes), 1  
 create\_list() (in module routes), 1

**E**

entry\_delete() (in module routes), 1  
 entry\_title\_save() (in module routes), 1

**G**

get\_id() (in module routes), 1  
 get\_pref() (in module routes), 1  
 getlist() (in module routes), 2

**H**

human\_readable\_time\_since() (in module routes), 2

**I**

index() (in module routes), 2  
 InvalidAPIUsage, 1  
 isLoggedIn() (in module routes), 2

**K**

key\_save() (in module routes), 2  
 key\_val\_save() (in module routes), 2  
 kvp\_delete() (in module routes), 2

**L**

list\_delete() (in module routes), 2  
 list\_save() (in module routes), 2  
 list\_title\_save() (in module routes), 3  
 login() (in module routes), 3  
 logout() (in module routes), 3

**M**

make\_list() (in module routes), 3  
 my\_lists\_interact() (in module routes), 3

**P**

perm\_list (in module routes), 3  
 permissions\_get() (in module routes), 3  
 permissions\_set() (in module routes), 3  
 preview\_list() (in module routes), 3  
 public\_level\_set() (in module routes), 4

**Q**

query\_cur\_perm() (in module routes), 4  
 query\_permission() (in module routes), 4

**R**

register() (in module routes), 4  
 remove\_from\_my\_lists() (in module routes), 4  
 Role (class in routes), 1  
 routes (module), 1

**S**

signin() (in module routes), 4  
 signup() (in module routes), 4

**U**

User (class in routes), 1  
 userlists() (in module routes), 4

**V**

value\_save() (in module routes), 4  
 vote() (in module routes), 4