BadgeKit Client Documentation

Release 0.5.1

Thomas Grenfell Smith

June 09, 2014

Contents

A low-level interface to the BadgeKit API.

The `BadgeKitAPI` object provides the ability to list, get, add, modify, etc., the objects in the BadgeKit REST API. The goal is to abstract away as much of the network communication as possible, while not including a lot of special data about the structure of the API methods. So, you will still need to know which methods can be called on which API endpoints, etc. **At this point, a list of available API endpoints** exists here.

As an example of translating between that list and this client, the list says you can

```
GET /systems/:slug/issuers/:slug/programs/:slug/badges/:slug/applications HTTP/1.1
```

With this client, you could say the following to hit that endpoint:

```
api.list('application', system=sys, issuer=iss, program=prog, badge=badge)
```

Notice that all the parameters are singular forms of the nouns. The order of the method parameters does not matter, they are pulled into order when the URL is constructed.

Note that these classes are listed as being part of `badgekit.api`, but you can import them straight from `badgekit`.

class `badgekit.api.`**`BadgeKitAPI`**(*baseurl*, *secret*, *key='master'*, *defaults=None*)

A class representing an interface with the BadgeKit API server.

Parameters

- **baseurl** – the URL of the badgekit-api server.
- **secret** – the client secret.
- **key** – the name of the client secret, for the server to see.
- **defaults** – a dict of default arguments, which can be overridden by actual arguments to the functions.

For the moment, the secret is just the same secret that is used between the two Node.js servers, badgekit-api and openbadges-badgekit. Look for it in the environment variables of the badgekit-api server - maybe a file called *.env*.

The `defaults` argument can be quite useful. If your code will only have to work with one badge system at a time - the badge system is in a configuration file, for instance - then you can set it once and then not have to pass extra info around to all your code that uses the api:

```
>>> bk = BadgeKitAPI('http://api.example.com/', 'secr3t', defaults={'system': 'mysystem'})
```

`create`(*kind*, *data*, ***kwargs*)

Create an object in the API.

Parameters

- **kind** – The kind of object to create - 'badge', 'issuer', 'instance', etc.
- **data** – The data fields of the object, as a dict.

```
>>> bk.create('badge', {'name': 'Super', ...}, system='mysystem')
{ ... }
```

The remaining keyword arguments specify the future location of the object.

Use this method to `POST` a URL that ends with a kind of object. For instance, the above code would post to `/systems/mysystem/badges` with `data` as the body of the request.

`delete`(***kwargs*)

Delete an object - not implemented yet

get(***kwargs*)
Retrieves some object from the API.

```
>>> bk.get(system='mysystem', badge='stupendous-badge')
{ ... }
```

The arguments should all be keywords, specifying the location of the object.

Use this method to `GET` a URL that ends with an identifier of an object, and you just want to get that one object - for example, the above code would hit `/systems/mysystem/badges/stupendous-badge`.

get_public_url(*url*)
GET a URL, and parse its JSON, checking for known errors. Useful for public URLs on the BadgeKit API server (e.g. assertions).

list(*kind*, ***kwargs*)
Lists objects present in some container or badge.

Parameters **kind** – The 'kind' of object to list - for example, 'badge' or 'system'.

```
>>> bk.list('badge', system='mysystem')
[ ... ]
```

The first argument is the kind of object that you would like to list. All other arguments should be keywords specifying the location of that object, for instance the system, issuer, and program.

Use this method to `GET` a URL that ends with the name of a 'kind' of object - for example, the above code would hit `/systems/mysystem/badges`.

ping()
Tests the server's availability - returns True if server is available, False otherwise.

require_server_version(*required_version*)
Require a certain version of the BadgeKit API Server.

The BadgeKit-API server is under rapid development. If you require particular features, it makes sense to check the server version so that you get "fail-early" behavior from your application. This method checks the supplied `required_version` against the server's version, parses them with `distutils.version.StrictVersion`, and compares them. If the server version is strictly less than `required_version`, a `ValueError` is raised with an informative error message.

server_version()
Returns the server's reported version as a string.

update(*data*, ***kwargs*)
Update an object - not implemented yet

exception `badgekit.api.`**RequestException**
There was an ambiguous exception that occurred while handling your request.

exception `badgekit.api.`**APIError**
Thrown for unexpected problems, maybe problems in this library, or invalid JSON from the server.

exception `badgekit.api.`**ResourceNotFound**(*resp_obj*, *request*)
Thrown for HTTP 404 when it is meaningful for the API

exception `badgekit.api.`**ResourceConflict**(*resp_obj*, *request*)
Thrown when POSTing an item that already exists

exception `badgekit.api.`**ValidationError**(*resp_obj*, *request*)
Thrown when creating an item with improper or missing fields

b