restnavigator Documentation

Release 1.0.3

Josh Kuhn

Mar 05, 2018

Contents

1 REST Navigator **3**

1.1 Contents . . . 3

1.2 How to use it . . . 4

1.3 Additional Topics . . . 7

1.4 Development . . . 10

2 Internal API **13**

3 Changelog **17**

3.1 Unreleased . . . 17

3.2 1.0 . . . 17

Python Module Index **19**

Contents

- *Welcome to restnavigator's documentation!*
- *REST Navigator*
  - *Contents*
  - *How to use it*
    - *Links*
    - *GET requests*
    - *Link relation docs*
    - *POST requests*
    - *Errors*
    - *Templated links*
    - *Authentication*
  - *Additional Topics*
    - *Identity Map*
    - *Iterating over a Navigator*
    - *Headers (Request vs. Response)*
    - *Bracket mini-language*
    - *Finding the right link*
    - *Default curie*
    - *Specifying an api name*
    - *Embedded documents*
  - *Development*
    - *Testing*
    - *Planned for the future*
    - *Contributors*
- *Internal API*
- *Changelog*
  - *Unreleased*
  - *1.0*

CHAPTER 1

REST Navigator

Note: this project is unmaintained! Feel free to fork it!

REST Navigator is a python library for interacting with hypermedia apis (REST level 3). Right now, it only supports HAL+JSON but it should be general enough to extend to other formats eventually. Its first goal is to make interacting with HAL hypermedia apis as painless as possible, while discouraging REST anti-patterns.

To install it, simply use pip:

```
$ pip install restnavigator
```

1.1 Contents

- *How to use it*
  - *Links*
  - *GET requests*
  - *Link relation docs*
  - *POST requests*
  - *Errors*
  - *Templated links*
  - *Authentication*
- *Additional Topics*
  - *Identity Map*
  - *Iterating over a Navigator*
  - *Headers (Request vs. Response)*

    - *Bracket mini-language*
    - *Finding the right link*
    - *Default curie*
    - *Specifying an api name*
    - *Embedded documents*
- *Development*
    - *Testing*
    - *Planned for the future*

1.2 How to use it

To begin interacting with a HAL api, you've got to create a HALNavigator that points to the api root. Ideally, in a hypermedia API, the root URL is the only URL that needs to be hardcoded in your application. All other URLs are obtained from the api responses themselves (think of your api client as 'clicking on links', rather than having the urls hardcoded).

As an example, we'll connect to the haltalk api.

```
>>> from restnavigator import Navigator
>>> N = Navigator.hal('http://haltalk.herokuapp.com/', default_curie="ht")
>>> N
HALNavigator(Haltalk)
```

1.2.1 Links

Usually, with the index (normally at the api root), you're most interested in the links. Let's look at those:

```
>>> N.links()
{u'ht:users': HALNavigator(Haltalk.users),
 u'ht:signup': HALNavigator(Haltalk.signup),
 u'ht:me': TemplatedThunk(Haltalk.users.{name}),
 u'ht:latest-posts': HALNavigator(Haltalk.posts.latest)}
```

(This may take a moment because asking for the links causes the HALNavigator to actually request the resource from the server).

Here we can see that the links are organized by their relation type (the key), and each key corresponds to a new HALNavigator that represents some other resource. Relation types are extremely important in restful apis: we need them to be able to determine what a link means in relation to the current resource, in a way that is automatable.

1.2.2 GET requests

In addition, the root has some state associated with it which you can get in two different ways:

```
>>> N() # cached state of resource (obtained when we looked at N.links)
{u'hint_1': u'You need an account to post stuff..',
 u'hint_2': u'Create one by POSTing via the ht:signup link..',
 u'hint_3': u'Click the orange buttons on the right to make POST requests..',
 u'hint_4': u'Click the green button to follow a link with a GET request..',
 u'hint_5': u'Click the book icon to read docs for the link relation.',
 u'welcome': u'Welcome to a haltalk server.'}
>>> N.fetch() # will refetch the resource from the server
{u'hint_1': u'You need an account to post stuff..',
 u'hint_2': u'Create one by POSTing via the ht:signup link..',
 u'hint_3': u'Click the orange buttons on the right to make POST requests..',
 u'hint_4': u'Click the green button to follow a link with a GET request..',
 u'hint_5': u'Click the book icon to read docs for the link relation.',
 u'welcome': u'Welcome to a haltalk server.'}
```

Calling a HALNavigator will execute a GET request against the resource and returns its value (which it will cache).

1.2.3 Link relation docs

Let's register a hal talk account. Unfortunately, we don't really know how to do that, so let's look at the documentation. The `ht:signup` link looks promising, let's check that:

```
>>> N.docsfor('ht:signup')
```

A browser will open to http://haltalk.herokuapp.com/rels/signup.

What? Popping up a browser from a library call? Yes, that's how rest_navigator rolls. The way we see it: docs are for humans, and while custom rel-types are URIs, they shouldn't automatically be dereferenced by a program that interacts with the api. So popping up a browser serves two purposes:

1. It allows easy access to the documentation at the time when you most need it: when you're mucking about in the command line trying to figure out how to interact with the api.
2. It reminds you not to try to automatically dereference the rel documentation and parse it in your application.

If you need a more robust way to browse the api and the documentation, HAL Browser is probably your best bet.

1.2.4 POST requests

The docs for `ht:signup` explain the format of the POST request to sign up. So let's actually sign up. Since we've set `"ht"` as our default curie, we can skip typing the curie for convenience. (Note: haltalk is a toy api for example purposes, don't ever send plaintext passwords over an unencrypted connection in a real app!):

```
>>> fred23 = N['signup'].create(
... {'username': 'fred23',
...  'password': 'hunter2',
...  'real_name': 'Fred 23'}
... )
>>> fred23
HALNavigator(Haltalk.users.fred23)
```

1.2.5 Errors

If the user name had already been in use, a 400 would have been returned from the haltalk api. rest_navigator follows the Zen of Python guideline "Errors should never pass silently". An exception would have been raised on a 400 or 500 status code. You can squelch this exception and just have the post call return a `HALNavigator` with a 400/500 status code if you want:

```
>>> dup_signup = N['ht:signup'].create({
...     'username': 'fred23',
...     'password': 'hunter2',
...     'real_name': 'Fred Wilson'
... }, raise_exc=False)
>>> dup_signup
OrphanHALNavigator(Haltalk.signup)  # 400!
>>> dup_signup.status
(400, 'Bad Request')
>>> dup_signup.state
{u"errors": {u"username": [u"is already taken"]}}
```

1.2.6 Templated links

Now that we've signed up, lets take a look at our profile. The link for a user's profile is a templated link, which restnavigator represents as a `PartialNavigator`. Similar to python's functools.partial, a `PartialNavigator` is an object that needs a few more arguments to give you a full navigator back. Despite its name, it can't talk to the network by itself. Its job is to to generate new navigators for you. You can see what variables it has by looking at its `.variables` attribute (its `__repr__` hints at this as well):

```
>>> N.links().keys()
['ht:latest-posts', 'ht:me', 'ht:users', 'ht:signup']
>>> N['ht:me']
PartialNavigator(Haltalk.users.{name})
>>> N['ht:me'].variables
set(['name'])
```

The documentation for the `ht:me` rel type should tell us how the name parameter is supposed to work, but in this case it's fairly obvious (plug in the username). Two provide the template parameters, just call it with keyword args:

```
>>> partial_me = N['ht:me']
>>> partial_me.template_uri
'http://haltalk.herokuapp.com/users/{name}'
>>> Fred = partial_me(name='fred23')
>>> Fred
HALNavigator('haltalk.users.fred23')
```

Now that we have a real navigator, we can fetch the resource:

```
>>> Fred()
{u'bio': None, u'real_name': u'Fred Wilson', u'username': u'fred23'}
```

1.2.7 Authentication

In order to post something to haltalk, we need to authenticate with our newly created account. HALNavigator allows any authentication method that requests supports (so OAuth etc). For basic auth (which haltalk uses), we can just pass a tuple.

```
>>> N.authenticate(('fred23', 'hunter2'))  # All subsequent calls are authenticated
```

This doesn't send anything to the server, it just sets the authentication details that we'll use on the next request. Other authentication methods may contact the server immediately.

Now we can put it all together to create a new post:

```
>>> N_post = N['me'](name='fred23')['posts'].create({'content': 'My first post'})
>>> N_post
HALNavigator(Haltalk.posts.523670eff0e6370002000001)
>>> N_post()
{'content': 'My first post', 'created_at': '2015-06-13T19:38:59+00:00'}
```

It is also possible to specify a custom requests Session object when creating a new navigator.

For example, if you want to talk to a OAuth2 protected api, simply pass an OAuth2 Session object that will be used for all requests done by HALNavigator:

```
>>> from requests_oauthlib import OAuth2Session
>>> oauth2_session = OAuth2Session(r'client_id', token='token')
>>> N = Navigator.hal('https://api.example.com', session=oauth2_session)
```

1.3 Additional Topics

1.3.1 Identity Map

You don't need to worry about inadvertently having two different navigators pointing to the same resource. rest_navigator will reuse the existing navigator instead of creating a new one

1.3.2 Iterating over a Navigator

If a resource has a link with the rel "next", the navigator for that resource can be used as a python iterator. It will automatically raise a StopIteration exception if a resource in the chain does not have a next link. This makes moving through paged resources really simple and pythonic:

```
post_navigator = fred['ht:posts']
for post in post_navigator:
    # the first post will be post_navigator itself
    print(post.state)
```

1.3.3 Headers (Request vs. Response)

HTTP response headers are available in `N.response.headers`

Headers that will be sent on each request can be obtained through the session:

```
>>> N.session.headers
# Cookies, etc
```

1.3.4 Bracket mini-language

The bracket (`[]`) operator on Navigators has a lot of power. As we saw earlier, the main use is to get a new Navigator from a link relation:

```
>>> N2 = N['curie:link_rel']
```

But, it can also go more than one link deep, which is equivalent to using multiple brackets in a row:

```
>>> N3 = N['curie:first_link', 'curie:second_link']
# equivalent to:
N3 = N['curie:first_link']['curie:second_link']
```

And of course, if you set a default curie, you can omit it:

```
>>> N3 = N['first_link', 'second_link']
```

Internally, this is completely equivalent to repeatedly applying the bracket operator, so you can even use it to jump over intermediate objects that aren't Navigators themselves:

```
>>> N['some-link', 3, 'another-link']
```

This would use the `some-link` link relation, select the third link from the list, and then follow `another-link` from that resource.

1.3.5 Finding the right link

Normally, you can chain together brackets to jump from one resource to another in one go:

```
>>> N['ht:widget']['ht:gadget']
```

This will return a Navigator for the `ht:widget` link relation and then immediately fetch the resource and return a Navigator for the `ht:gadget` link relation. This works great if you have only one link per relation, but HAL allows multiple links per relation. Say for instance we have some links like the following:

When we go to get the `ht:some_rel`, we'll get multiple results:

```
>>> N['ht:some_rel']
[HALNavigator(api.widget[1]),
 HALNavigator(api.widget[2]),
 HALNavigator(api.gadget[1])]
```

How do we know which one is the one we want? The HAL spec says links with the same rel can be disambiguated by the `name` link property:

```
>>> N.links['ht:some_rel'].get_by('name', 'gadget1')
HALNavigator(api.gadget[1])
>>> N.links['ht:some_rel'].named('gadget1')  # same as previous
HALNavigator(api.gadget[1])
```

We could also use other properties to slice and dice the list:

```
>>> N.links['ht:some_rel'].get_by('profile', 'gadget')
HALNavigator(api.gadget[1])
>>> N.links['ht:some_rel'].getall_by('profile', 'widget')
[HALNavigator(api.widget[1]), HALNavigator(api.widget[2])]
```

This works for any property on links, not just the standard HAL properties.

1.3.6 Default curie

You may specify a default curie when creating your Navigator:

```
>>> N = HALNavigator('http://haltalk.herokuapp.com', curie='ht')
```

Now, when you follow links, you may leave off the default curie if you want:

```
>>> N.links
{'ht:users': [HALNavigator(Haltalk.users)],
 'ht:signup': [HALNavigator(Haltalk.signup)],
 'ht:me': [HALNavigator(Haltalk.users.{name})],
 'ht:latest-posts': [HALNavigator(Haltalk.posts.latest)]
}
>>> N['ht:users']
HALNavigator(Haltalk.users)
>>> N['users']
HALNavigator(Haltalk.users)
```

The only exception is where the key being supplied is a IANA registered link relation, and there is a conflict (hint: this should be quite rare):

```
>>> N.links
{'ht:next': HALNavigator(Haltalk.unregistered),
  'next': HALNavigator(Haltalk.registered)}
>>> N['next']
HALNavigator(Haltalk.registered)
```

1.3.7 Specifying an api name

Sometimes the automatic api naming guesses poorly. If you'd like to override the default name, you can specify it when creating the navigator:

```
>>> N = Navigator.hal('http://api.example.com', apiname='MySpecialAPI')
HALNavigator(MySpecialAPI)
```

1.3.8 Embedded documents

In rest_navigator, embedded documents are treated transparently. This means that in many cases you don't need to worry about whether a document is embedded or whether it's just linked.

As an example, assume we have a resource like the following:

```
{
  "_links": {
     ...
     "xx:yams": {
        "href": "/yams"
     }
     ...
  },
  "_embedded": {
     "xx:pickles": {
       "_links": {
         "self": {"href": "/pickles"}
       },
       "state": "A pickle"
     }
  }
  ...
}
```

From here, you would access both the `yams` and the `pickles` resource with normal bracket syntax:

```
>>> Yams = N['xx:yams']
>>> Pickles = N['xx:pickles']
```

The only difference here is that `Yams` hasn't been fetched yet, while `Pickles` is considered "resolved" already because we got it as an embedded document.

```
>>> Yams.resolved
False
>>> Yams.state # None
>>> Pickles.resolved
True
>>> Pickles.state
{'state': 'A pickle'}
```

If an embedded document has a self link, you can treat it just like you would any other resource. So if you want to refresh the resource, it's as easy as:

```
>>> Pickles.fetch()
```

This will fetch the current state of the resource from the uri in its self link, even if you've never directly requested that uri before. If an embedded resource doesn't have a self link, it will be an `OrphanNavigator` with the parent set to the resource it was embedded in.

Of course, if you need to directly distinguish between linked resources and embedded resources, there is an out:

```
>>> N.embedded()
{'xx:pickles': HALNavigator(api.pickles)
>>> N.links()
{'xx:yams': HALNavigator(api.yams)
```

However, when using the `in` operator, it will look in both for a key you're interested in:

```
>>> 'yams' in N  # default curie is taken into account!
True
>>> 'xx:yams in N
True
>>> 'xx:pickles' in N
True
```

1.4 Development

1.4.1 Testing

To run tests, first install the pytest framework:

```
$ pip install -U pytest
```

To run tests, execute following from the root of the source directory:

```
$ py.test
```

1.4.2 Planned for the future

- Ability to add hooks for different types, rels and profiles. If a link has one of these properties, it will call your hook when doing a server call.
- Since HAL doesn't specify what content type POSTs, PUTs, and PATCHes need to have, you can specify the hooks based on what the server will accept. This can trigger off either the rel type of the link, or rest navigator can do content negotiation over HTTP with the server directly to see what content types that resource will accept.

1.4.3 Contributors

Thanks very much to rest navigator's contributors:

- dudycooly
- bubenkoff
- bbsgfalconer

CHAPTER 2

Internal API

A library to allow navigating rest apis easy.

class restnavigator.halnav.**APICore**(*root*, *nav_class*, *apiname=None*, *default_curie=None*, *session=None*, *id_map=None*)

Shared data between navigators from a single api.

This should contain all state that is generally maintained from one navigator to the next.

authenticate(*auth*)

Sets the authentication for future requests to the api

cache(*link*, *nav*)

Stores a navigator in the identity map for the current api. Can take a link or a bare uri

get_cached(*link*, *default=None*)

Retrieves a cached navigator from the id_map.

Either a Link object or a bare uri string may be passed in.

is_cached(*link*)

Returns whether the current navigator is cached. Intended to be overwritten and customized by subclasses.

class restnavigator.halnav.**HALNavigator**(*link*, *core*, *response=None*, *state=None*, *curies=None*, *_links=None*, *_embedded=None*)

The main navigation entity

create(*body=None*, *raise_exc=True*, *headers=None*, ***kwargs*)

Performs an HTTP POST to the server, to create a subordinate resource. Returns a new HALNavigator representing that resource.

body may either be a string or a dictionary representing json *headers* are additional headers to send in the request

delete(*raise_exc=True*, *headers=None*, *files=None*)

Performs an HTTP DELETE to the server, to delete resource(s).

headers are additional headers to send in the request

fetch(*raise_exc=True*)
Performs a GET request to the uri of this navigator

patch(*body*, *raise_exc=True*, *headers=False*, *files=None*)
Performs an HTTP PATCH to the server. This is a non-idempotent call that may update all or a portion of the resource this navigator is pointing to. The format of the patch body is up to implementations.

body may either be a string or a dictionary representing json *headers* are additional headers to send in the request

upsert(*body*, *raise_exc=True*, *headers=False*, *files=None*)
Performs an HTTP PUT to the server. This is an idempotent call that will create the resource this navigator is pointing to, or will update it if it already exists.

body may either be a string or a dictionary representing json *headers* are additional headers to send in the request

class restnavigator.halnav.**HALNavigatorBase**(*link*, *core*, *response=None*, *state=None*, *curies=None*, *_links=None*, *_embedded=None*)
Base class for navigation objects

authenticate(*auth*)
Authenticate with the api

docsfor(*rel*)
Obtains the documentation for a link relation. Opens in a webbrowser window

embedded()
Returns a dictionary of navigators representing embedded documents in the current resource. If the navigators have self links they can be fetched as well.

links()
Returns a dictionary of navigators from the current resource. Fetches the resource if necessary.

class restnavigator.halnav.**Link**(*uri*, *properties=None*)
Represents a HAL link. Does not store the link relation

relative_uri(*root*)
Returns the link of the current uri compared against an api root

class restnavigator.halnav.**Navigator**
A factory for other navigators. Makes creating them more convenient

static hal(*root*, *apiname=None*, *default_curie=None*, *auth=None*, *headers=None*, *session=None*)
Create a HALNavigator

class restnavigator.halnav.**OrphanHALNavigator**(*link*, *core*, *response=None*, *state=None*, *curies=None*, *_links=None*, *parent=None*)
A Special navigator that is the result of a non-GET

This navigator cannot be fetched or created, but has a special property called *.parent* that refers to the navigator this one was created from. If the result is a HAL document, it will be populated properly

class restnavigator.halnav.**PartialNavigator**(*link*, *core=None*)
A lazy representation of a navigator. Expands to a full navigator when template arguments are given by calling it.

expand_link(***kwargs*)
Expands with the given arguments and returns a new untemplated Link object

expand_uri(***kwargs*)
Returns the template uri expanded with the current arguments

variables
Returns a set of the template variables in this templated link

exception `restnavigator.exc.`**HALNavigatorError**(*message*, *nav=None*, *status=None*, *response=None*)
Raised when a response is an error

Has all of the attributes of a normal HALNavigator. The error body can be returned by examining response.body

exception `restnavigator.exc.`**NoResponseError**
Raised when accessing a field of a navigator that has not fetched a response yet

exception `restnavigator.exc.`**OffTheRailsException**(*traversal*, *index*, *intermediates*, *e*)
Raised when a traversal specified to __getitem__ cannot be satisfied

exception `restnavigator.exc.`**UnexpectedlyNotJSON**(*uri*, *response*)
Raised when a non-json parseable resource is gotten

exception `restnavigator.exc.`**WileECoyoteException**
Raised when a url has a bad scheme

exception `restnavigator.exc.`**ZachMorrisException**
Raised when a url has too many schemes

class `restnavigator.utils.`**CurieDict**(*default_curie*, *d*)
dict subclass that allows specifying a default curie. This enables multiple ways to access an item

class `restnavigator.utils.`**LinkList**(*items=None*)
A list subclass that offers different ways of grabbing the values based on various metadata stored for each entry in the dictionary.

Note: Removing items from this list isn't really the point, so no attempt has been made to make this convenient. Deleting items will not remove them from the list's metadata.

append_with(*obj*, ***properties*)
Add an item to the dictionary with the given metadata properties

get_by(*prop*, *val*, *raise_exc=False*)
Retrieve an item from the dictionary with the given metadata properties. If there is no such item, None will be returned, if there are multiple such items, the first will be returned.

getall_by(*prop*, *val*)
Retrieves all items from the dictionary with the given metadata

named(*name*)
Returns .get_by('name', name)

serialize
alias of `unicode`

`restnavigator.utils.`**fix_scheme**(*url*)
Prepends the http:// scheme if necessary to a url. Fails if a scheme other than http is used

`restnavigator.utils.`**getpath**(*d*, *json_path*, *default=None*, *sep='.'*)
Gets a value nested in dictionaries containing dictionaries. Returns the default if any key in the path doesn't exist.

`restnavigator.utils.`**getstate**(*d*)
Deep copies a dict, and returns it without the keys _links and _embedded

`restnavigator.utils.`**`namify`**`(`*root_uri*`)`
: Turns a root uri into a less noisy representation that will probably make sense in most circumstances. Used by Navigator's __repr__, but can be overridden if the Navigator is created with a 'name' parameter.

`restnavigator.utils.`**`normalize_getitem_args`**`(`*args*`)`
: Turns the arguments to __getitem__ magic methods into a uniform list of tuples and strings

`restnavigator.utils.`**`objectify_uri`**`(`*relative_uri*`)`
: Converts uris from path syntax to a json-like object syntax. In addition, url escaped characters are unescaped, but non-ascii characters a romanized using the unidecode library.

  Examples: "/blog/3/comments" becomes "blog[3].comments" "car/engine/piston" becomes "car.engine.piston"

`restnavigator.utils.`**`parse_media_type`**`(`*media_type*`)`
: Returns type, subtype, parameter tuple from an http media_type. Can be applied to the 'Accept' or 'Content-Type' http header fields.

CHAPTER 3

Changelog

3.1 Unreleased

- TBD

3.2 1.0

- Embedded support
- Ability to specify default curies
- Resources with no URL are now represented by a special Navigator type called OrphanNavigators
- IP addresses can be used in the url (@dudycooly)
- All tests pass in python 2.6 -> 3.4 (@bubenkoff), and travis now runs tox to ensure they stay that way
- Support the DELETE, and PATCH methods
- posts allow an empty body (@bbsgfalconer)
- Much improved content negotiation (@bbsgfalconer)
- **There was also a major refactoring that changed how Navigators are created and internally cleaned up a**
  lot of really messy code.

Python Module Index

r

restnavigator, 13
restnavigator.exc, 15
restnavigator.halnav, 13
restnavigator.registry, 15
restnavigator.utils, 15

Index

A

APICore (class in restnavigator.halnav), 13
append_with() (restnavigator.utils.LinkList method), 15
authenticate() (restnavigator.halnav.APICore method), 13
authenticate() (restnavigator.halnav.HALNavigatorBase method), 14

C

cache() (restnavigator.halnav.APICore method), 13
create() (restnavigator.halnav.HALNavigator method), 13
CurieDict (class in restnavigator.utils), 15

D

delete() (restnavigator.halnav.HALNavigator method), 13
docsfor() (restnavigator.halnav.HALNavigatorBase method), 14

E

embedded() (restnavigator.halnav.HALNavigatorBase method), 14
expand_link() (restnavigator.halnav.PartialNavigator method), 14
expand_uri() (restnavigator.halnav.PartialNavigator method), 14

F

fetch() (restnavigator.halnav.HALNavigator method), 13
fix_scheme() (in module restnavigator.utils), 15

G

get_by() (restnavigator.utils.LinkList method), 15
get_cached() (restnavigator.halnav.APICore method), 13
getall_by() (restnavigator.utils.LinkList method), 15
getpath() (in module restnavigator.utils), 15
getstate() (in module restnavigator.utils), 15

H

hal() (restnavigator.halnav.Navigator static method), 14
HALNavigator (class in restnavigator.halnav), 13
HALNavigatorBase (class in restnavigator.halnav), 14
HALNavigatorError, 15

I

is_cached() (restnavigator.halnav.APICore method), 13

L

Link (class in restnavigator.halnav), 14
LinkList (class in restnavigator.utils), 15
links() (restnavigator.halnav.HALNavigatorBase method), 14

N

named() (restnavigator.utils.LinkList method), 15
namify() (in module restnavigator.utils), 15
Navigator (class in restnavigator.halnav), 14
NoResponseError, 15
normalize_getitem_args() (in module restnavigator.utils), 16

O

objectify_uri() (in module restnavigator.utils), 16
OffTheRailsException, 15
OrphanHALNavigator (class in restnavigator.halnav), 14

P

parse_media_type() (in module restnavigator.utils), 16
PartialNavigator (class in restnavigator.halnav), 14
patch() (restnavigator.halnav.HALNavigator method), 14

R

relative_uri() (restnavigator.halnav.Link method), 14
restnavigator (module), 13
restnavigator.exc (module), 15
restnavigator.halnav (module), 13
restnavigator.registry (module), 15
restnavigator.utils (module), 15

S

serialize (restnavigator.utils.LinkList attribute), 15

U

UnexpectedlyNotJSON, 15
upsert() (restnavigator.halnav.HALNavigator method), 14

V

variables (restnavigator.halnav.PartialNavigator attribute), 15

W

WileECoyoteException, 15

Z

ZachMorrisException, 15