URLObject Documentation

Release 2.3.4

Zachary Voase

February 18, 2013

CONTENTS

URLObject is a utility class for manipulating URLs. The latest incarnation of this library builds upon the ideas of its predecessor, but aims for a clearer API, focusing on proper method names over operator overrides. It's also being developed from the ground up in a test-driven manner, and has full Sphinx documentation.

If you're new to this library, take a look at the *quickstart*. If you're just here for reference purposes, see the *API documentation*.

CHAPTER

ONE

INSTALLATION

Install using `pip` from pypi. There are no dependencies, and the package is pure Python, tested on CPythons v2.5–3.3.

CHAPTER

TWO

CONTENTS

2.1 Quickstart

Note: All doctests in this documentation use Python 3.3 syntax.

```
>>> from urlobject import URLObject
```

Create a URLObject with a string representing a URL. `URLObject` is a regular subclass of `unicode` (or `str` if you're using Python 3), it just has several properties and methods which make it easier to manipulate URLs. All the basic slots from urlsplit are there:

```
>>> url = URLObject("https://github.com/zacharyvoase/urlobject?spam=eggs#foo")
>>> print(url)
https://github.com/zacharyvoase/urlobject?spam=eggs#foo
>>> print(url.scheme)
https
>>> print(url.netloc)
github.com
>>> print(url.hostname)
github.com
>>> (url.username, url.password)
(None, None)
>>> print(url.port)
None
>>> url.default_port
443
>>> print(url.path)
/zacharyvoase/urlobject
>>> print(url.query)
spam=eggs
>>> print(url.fragment)
foo
```

You can replace any of these slots using a `with_*()` method. Remember that, because `unicode` (and therefore `URLObject`) is immutable, these methods all return new URLs:

```
>>> print(url.with_scheme('http'))
http://github.com/zacharyvoase/urlobject?spam=eggs#foo
>>> print(url.with_netloc('example.com'))
https://example.com/zacharyvoase/urlobject?spam=eggs#foo
>>> print(url.with_auth('alice', '1234'))
https://alice:1234@github.com/zacharyvoase/urlobject?spam=eggs#foo
```

```
>>> print(url.with_path('/some_page'))
https://github.com/some_page?spam=eggs#foo
>>> print(url.with_query('funtimes=yay'))
https://github.com/zacharyvoase/urlobject?funtimes=yay#foo
>>> print(url.with_fragment('example'))
https://github.com/zacharyvoase/urlobject?spam=eggs#example
```

For the query and fragment, `without_` methods also exist:

```
>>> print(url.without_query())
https://github.com/zacharyvoase/urlobject#foo
>>> print(url.without_fragment())
https://github.com/zacharyvoase/urlobject?spam=eggs
```

2.1.1 Relative URL Resolution

You can resolve relative URLs against a URLObject using `relative()`:

```
>>> print(url.relative('another-project'))
https://github.com/zacharyvoase/another-project
>>> print(url.relative('?different-query-string'))
https://github.com/zacharyvoase/urlobject?different-query-string
>>> print(url.relative('#frag'))
https://github.com/zacharyvoase/urlobject?spam=eggs#frag
```

Absolute URLs will just be returned as-is:

```
>>> print(url.relative('http://example.com/foo'))
http://example.com/foo
```

And you can specify as much or as little of the new URL as you like:

```
>>> print(url.relative('//example.com/foo'))
https://example.com/foo
>>> print(url.relative('/dvxhouse/intessa'))
https://github.com/dvxhouse/intessa
>>> print(url.relative('/dvxhouse/intessa?foo=bar'))
https://github.com/dvxhouse/intessa?foo=bar
>>> print(url.relative('/dvxhouse/intessa?foo=bar#baz'))
https://github.com/dvxhouse/intessa?foo=bar#baz
```

2.1.2 Path

The `path` property is an instance of `URLPath`, which has several methods and properties for manipulating the path string:

```
>>> print(url.path)
/zacharyvoase/urlobject
>>> print(url.path.parent)
/zacharyvoase/
>>> print(url.path.segments)
('zacharyvoase', 'urlobject')
>>> print(url.path.add_segment('subnode'))
/zacharyvoase/urlobject/subnode
>>> print(url.path.root)
/
```

Some of these are aliased on the URL itself:

```
>>> print(url.parent)
https://github.com/zacharyvoase/?spam=eggs#foo
>>> print(url.add_path_segment('subnode'))
https://github.com/zacharyvoase/urlobject/subnode?spam=eggs#foo
>>> print(url.add_path('tree/urlobject2'))
https://github.com/zacharyvoase/urlobject/tree/urlobject2?spam=eggs#foo
>>> print(url.root)
https://github.com/?spam=eggs#foo
```

2.1.3 Query string

The `query` property is an instance of `QueryString`, so you can access sub-attributes of that with richer representations of the query string:

```
>>> print(url.query)
spam=eggs
>>> url.query.list  # aliased as url.query_list
[('spam', 'eggs')]
>>> url.query.dict  # aliased as url.query_dict
{'spam': 'eggs'}
>>> url.query.multi_dict  # aliased as url.query_multi_dict
{'spam': ['eggs']}
```

Modifying the query string is easy, too. You can 'add' or 'set' parameters: any method beginning with `add_` will allow you to use the same parameter name multiple times in the query string; methods beginning with `set_` will only allow one value for a given parameter name. Don't forget that each method will return a *new* `QueryString` instance, unattached to the original URL:

```
>>> print(url.query.add_param('spam', 'ham'))
spam=eggs&spam=ham
>>> print(url.query.set_param('spam', 'ham'))
spam=ham
>>> print(url.query.add_params({'spam': 'ham', 'foo': 'bar'}))
spam=eggs&foo=bar&spam=ham
>>> print(url.query.set_params({'spam': 'ham', 'foo': 'bar'}))
foo=bar&spam=ham
```

Delete parameters with `del_param()` and `del_params()`. These will remove any and all appearances of the requested parameter name from the query string, returning a new query string:

```
>>> print(url.query.del_param('spam')) # Result is empty

>>> print(url.query.add_params({'foo': 'bar', 'baz': 'blah'}).del_params(['spam', 'foo']))
baz=blah
```

Again, some of these methods are aliased on the `URLObject` directly:

```
>>> print(url.add_query_param('spam', 'ham'))
https://github.com/zacharyvoase/urlobject?spam=eggs&spam=ham#foo
>>> print(url.set_query_param('spam', 'ham'))
https://github.com/zacharyvoase/urlobject?spam=ham#foo
>>> print(url.del_query_param('spam'))
https://github.com/zacharyvoase/urlobject#foo
```

2.1.4 Next Steps

Check out the *API documentation* for a detailed description of all the properties and methods available on `URLObject`.

2.2 API

Note: All doctests in this documentation use Python 3.3 syntax.

---

class `urlobject.`**`URLObject`**

A URL.

This class contains properties and methods for accessing and modifying the constituent components of a URL. `URLObject` instances are immutable, as they derive from the built-in `unicode`, and therefore all methods return *new* objects; you need to consider this when using `URLObject` in your own code.

```
>>> from urlobject import URLObject
>>> u = URLObject("http://www.google.com/")
>>> print(u)
http://www.google.com/
```

URL objects feature properties for directly accessing different parts of the URL: `scheme`, `netloc`, `username`, `password`, `hostname`, `port`, `path`, `query` and `fragment`.

All of these have a `with_*` method for adding/replacing them, and some have a `without_*` method for removing them altogether. The query string and path also have a variety of methods for doing more fine-grained inspection and manipulation.

`scheme`

This URL's scheme.

```
>>> print(URLObject("http://www.google.com").scheme)
http
```

`with_scheme`(*scheme*)

Add or replace this URL's `scheme`.

```
>>> print(URLObject("http://www.google.com").with_scheme("ftp"))
ftp://www.google.com
>>> print(URLObject("//www.google.com").with_scheme("https"))
https://www.google.com
```

`netloc`

The full network location of this URL.

This value incorporates `username`, `password`, `hostname` and `port`.

```
>>> print(URLObject("http://user:pass@www.google.com").netloc)
user:pass@www.google.com
```

`with_netloc`(*netloc*)

Add or replace this URL's `netloc`.

```
>>> print(URLObject("http://www.google.com/a/b/c").with_netloc("www.amazon.com"))
http://www.amazon.com/a/b/c
```

username
This URL's username, if any.

```
>>> print(URLObject("http://user@www.google.com").username)
user
>>> print(URLObject("http://www.google.com").username)
None
```

with_username(*username*)
Add or replace this URL's `username`.

```
>>> print(URLObject("http://user@www.google.com").with_username("user2"))
http://user2@www.google.com
```

without_username()
Remove this URL's `username`.

```
>>> print(URLObject("http://user@www.google.com/").without_username())
http://www.google.com/
```

password
This URL's password, if any.

```
>>> print(URLObject("http://user:somepassword@www.google.com").password)
somepassword
>>> print(URLObject("http://user@www.google.com").password)
None
```

with_password(*password*)
Add or replace this URL's `password`.

```
>>> print(URLObject("http://user:somepassword@www.google.com").with_password("passwd"))
http://user:passwd@www.google.com
```

without_password()
Remove this URL's `password`.

```
>>> print(URLObject("http://user:pwd@www.google.com").without_password())
http://user@www.google.com
```

hostname
This URL's hostname.

```
>>> print(URLObject("http://www.google.com").hostname)
www.google.com
```

with_hostname(*hostname*)
Add or replace this URL's `hostname`.

```
>>> print(URLObject("http://www.google.com/a/b/c").with_hostname("cdn.amazon.com"))
http://cdn.amazon.com/a/b/c
```

port
This URL's port number, or `None`.

```
>>> URLObject("http://www.google.com:8080").port
8080
>>> print(URLObject("http://www.google.com").port)
None
```

default_port
The destination port number for this URL.

If no port number is explicitly given in the URL, this will return the default port number for the scheme if one is known, or `None`. The mapping of schemes to default ports is defined in `urlobject.ports.DEFAULT_PORTS`.

For URLs *with* explicit port numbers, this just returns the value of `port`.

```
>>> URLObject("https://www.google.com").default_port
443
>>> URLObject("http://www.google.com").default_port
80
>>> URLObject("http://www.google.com:126").default_port
126
```

with_port(*port*)
Add or replace this URL's `port`.

```
>>> print(URLObject("http://www.google.com/a/b/c").with_port(8080))
http://www.google.com:8080/a/b/c
```

without_port()
Remove this URL's `port`.

```
>>> print(URLObject("http://www.google.com:8080/a/b/c").without_port())
http://www.google.com/a/b/c
```

auth
The username and password of this URL as a 2-tuple.

```
>>> URLObject("http://user:password@www.google.com").auth
('user', 'password')
>>> URLObject("http://user@www.google.com").auth
('user', None)
>>> URLObject("http://www.google.com").auth
(None, None)
```

with_auth(**auth*)
Add or replace this URL's `username` and `password`.

With two arguments, this method adds/replaces both username and password. With one argument, it adds/replaces the username and removes any password.

```
>>> print(URLObject("http://user:password@www.google.com").with_auth("otheruser", "otherpass
http://otheruser:otherpassword@www.google.com
>>> print(URLObject("http://www.google.com").with_auth("user"))
http://user@www.google.com
```

without_auth()
Remove any `username` and `password` on this URL.

```
>>> print(URLObject("http://user:password@www.google.com/a/b/c").without_auth())
http://www.google.com/a/b/c
```

path
This URL's path.

```
>>> print(URLObject("http://www.google.com/a/b/c").path)
/a/b/c
>>> print(URLObject("http://www.google.com").path)
```

with_path(*path*)
Add or replace this URL's `path`.

```
>>> print(URLObject("http://www.google.com/a/b/c").with_path("c/b/a"))
http://www.google.com/c/b/a
```

root
The root node of this URL.

This is just a synonym for `url.with_path('/')`.

```
>>> print(URLObject("http://www.google.com/a/b/c").root)
http://www.google.com/
```

parent
The direct parent node of this URL.

```
>>> print(URLObject("http://www.google.com/a/b/c").parent)
http://www.google.com/a/b/
>>> print(URLObject("http://www.google.com/a/b/").parent)
http://www.google.com/a/
```

is_leaf
Whether this URL's `path` is a leaf node or not.

A leaf node is simply one without a trailing slash. Leaf-ness affects things like relative URL resolution (c.f. `relative()`) and server-side routing.

```
>>> URLObject("http://www.google.com/a/b/c").is_leaf
True
>>> URLObject('http://www.google.com/a/').is_leaf
False
>>> URLObject('http://www.google.com').is_leaf
False
```

add_path_segment(*segment*)

```
>>> print(URLObject("http://www.google.com").add_path_segment("a"))
http://www.google.com/a
```

add_path(*partial_path*)

```
>>> print(URLObject("http://www.google.com").add_path("a/b/c"))
http://www.google.com/a/b/c
```

query
This URL's query string.

```
>>> print(URLObject("http://www.google.com").query)

>>> print(URLObject("http://www.google.com?a=b").query)
a=b
```

with_query(*query*)
Add or replace this URL's `query` string.

```
>>> print(URLObject("http://www.google.com").with_query("a=b"))
http://www.google.com?a=b
```

without_query()
Remove this URL's query string.

```
>>> print(URLObject("http://www.google.com?a=b&c=d").without_query())
http://www.google.com
```

query_list
This URL's query as a list of name/value pairs.

This attribute is read-only. Changes you make to the list will not propagate back to the URL.

```
>>> URLObject("http://www.google.com?a=b&c=d").query_list
[('a', 'b'), ('c', 'd')]
```

query_dict
This URL's query as a dict mapping names to values.

Each name will have only its last value associated with it. For all the values for a given key, see query_multi_dict.

```
>>> dictsort(URLObject("http://www.google.com?a=b&c=d").query_dict)
{'a': 'b', 'c': 'd'}
>>> dictsort(URLObject("http://www.google.com?a=b&a=c").query_dict)
{'a': 'c'}
```

query_multi_dict
This URL's query as a dict mapping names to lists of values.

All values associated with a given name will be represented, in order, in that name's list.

```
>>> dictsort(URLObject("http://www.google.com?a=b&c=d").query_multi_dict)
{'a': ['b'], 'c': ['d']}
>>> dictsort(URLObject("http://www.google.com?a=b&a=c").query_multi_dict)
{'a': ['b', 'c']}
```

add_query_param(*name*, *value*)
Add a single query parameter.

You can add several query parameters with the same name to a URL.

```
>>> print(URLObject("http://www.google.com").add_query_param("a", "b"))
http://www.google.com?a=b
>>> print(URLObject("http://www.google.com").add_query_param("a", "b").add_query_param("a",
http://www.google.com?a=b&a=c
```

add_query_params(**args*, ***kwargs*)
Add multiple query parameters.

Accepts anything you would normally pass to dict(): iterables of name/value pairs, keyword arguments and dictionary objects.

```
>>> print(URLObject("http://www.google.com").add_query_params([('a', 'b'), ('c', 'd')]))
http://www.google.com?a=b&c=d
>>> print(URLObject("http://www.google.com").add_query_params(a="b"))
http://www.google.com?a=b
```

set_query_param(*name*, *value*)
Set a single query parameter, overriding it if it exists already.

```
>>> print(URLObject("http://www.google.com?a=b&c=d").set_query_param("a", "z"))
http://www.google.com?c=d&a=z
```

set_query_params(**args*, ***kwargs*)
Set query parameters, overriding existing ones.

Accepts anything you would normally pass to `dict()`: iterables of name/value pairs, keyword arguments and dictionary objects.

```
>>> print(URLObject("http://www.google.com?a=b&c=d").set_query_params([('a', 'z'), ('d', 'e'
http://www.google.com?c=d&a=z&d=e
>>> print(URLObject("http://www.google.com?a=b").set_query_params(a="z"))
http://www.google.com?a=z
```

del_query_param(*name*)
Remove any and all query parameters with the given name from the URL.

```
>>> print(URLObject("http://www.google.com?a=b&c=d&c=e").del_query_param("c"))
http://www.google.com?a=b
```

del_query_params(*params*)
Remove multiple query params from the URL.

```
>>> print(URLObject("http://www.google.com?a=b&c=d&d=e").del_query_params(["c", "d"]))
http://www.google.com?a=b
```

fragment
This URL's fragment.

```
>>> print(URLObject("http://www.google.com/a/b/c#fragment").fragment)
fragment
```

with_fragment(*fragment*)
Add or replace this URL's `fragment`.

```
>>> print(URLObject("http://www.google.com/a/b/c#fragment").with_fragment("new_fragment"))
http://www.google.com/a/b/c#new_fragment
```

without_fragment()
Remove this URL's `fragment`.

```
>>> print(URLObject("http://www.google.com/a/b/c#fragment").without_fragment())
http://www.google.com/a/b/c
```

relative(*other*)
Resolve another URL relative to this one.

For example, if you have a browser currently pointing to `http://www.google.com/a/b/c/`, then an HTML element like `<a href="../d/e/f">` would resolve to `http://www.google.com/a/b/d/e/f` using this function.

```
>>> print(URLObject("http://www.google.com/a/b/c/").relative("../d/e/f"))
http://www.google.com/a/b/d/e/f
```

CHAPTER

THREE

LICENSE

This is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

For more information, please refer to http://unlicense.org/

CHAPTER

FOUR

CREDITS

This library bundles six, which is licensed as follows:

> Copyright (c) 2010-2012 Benjamin Peterson
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Many thanks go to Aron Griffis for porting this library to Python 3, and to vmalloc for work on the comprehensive API documentation and Sphinx setup.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*