hyperlink Documentation

Release 19.0.0

Mahmoud Hashemi

March 27, 2020

Contents

Cool URLs that don't change.

Hyperlink provides a pure-Python implementation of immutable URLs. Based on RFC 3986 and RFC 3987, the Hyperlink URL balances simplicity and correctness for both *URIs and IRIs*.

Hyperlink is tested against Python 2.7, 3.4, 3.5, 3.6, 3.7, 3.8, and PyPy.

For an introduction to the hyperlink library, its background, and URLs in general, see this talk from PyConWeb 2017 (and the accompanying slides).

CHAPTER 1

Installation and Integration

Hyperlink is a pure-Python package and only depends on the standard library. The easiest way to install is with pip:

```
pip install hyperlink
```

Then, URLs are just an import away:

```
from hyperlink import URL

url = URL.from_text(u'http://github.com/python-hyper/hyperlink?utm_source=readthedocs
↪')

better_url = url.replace(scheme=u'https', port=443)
org_url = better_url.click(u'.')

print(org_url.to_text())
# prints: https://github.com/python-hyper/

print(better_url.get(u'utm_source'))
# prints: readthedocs
```

See *the API docs* for more usage examples.

CHAPTER 2

Gaps

Found something missing in hyperlink? Pull Requests and Issues are welcome!

CHAPTER 3

Table of Contents

3.1 Hyperlink Design

The URL is a nuanced format with a long history. Suitably, a lot of work has gone into translating the standards, RFC 3986 and RFC 3987, into a Pythonic interface. Hyperlink's design strikes a unique balance of correctness and usability.

3.1.1 A Tale of Two Representations

The URL is a powerful construct, designed to be used by both humans and computers.

This dual purpose has resulted in two canonical representations: the URI and the IRI.

Even though the W3C themselves have recognized the confusion this can cause, Hyperlink's URL makes the distinction quite natural. Simply:

- **URI**: Fully-encoded, ASCII-only, suitable for network transfer
- **IRI**: Fully-decoded, Unicode-friendly, suitable for display (e.g., in a browser bar)

We can use Hyperlink to very easily demonstrate the difference:

```
>>> url = URL.from_text('http://example.com/café')
>>> url.to_uri().to_text()
u'http://example.com/caf%C3%A9'
```

We construct a URL from text containing Unicode (`é`), then transform it using `to_uri()`. This results in ASCII-only percent-encoding familiar to all web developers, and a common characteristic of URIs.

Still, Hyperlink's distinction between URIs and IRIs is pragmatic, and only limited to output. Input can contain *any mix* of percent encoding and Unicode, without issue:

```
>>> url = URL.from_text("http://example.com/caf%C3%A9 au lait/s'il vous plaît!")
>>> print(url.to_iri().to_text())
```

(continues on next page)

(continued from previous page)

```
http://example.com/café au lait/s'il vous plaît!
>>> print(url.to_uri().to_text())
http://example.com/caf%C3%A9%20au%20lait/s'il%20vous%20pla%C3%AEt!
```

Note that even when a URI and IRI point to the same resource, they will often be different URLs:

```
>>> url.to_uri() == url.to_iri()
False
```

And with that caveat out of the way, you're qualified to correct other people (and their code) on the nuances of URI vs IRI.

3.1.2 Immutability

Hyperlink's URL is notable for being an immutable representation. Once constructed, instances are not changed. Methods like `click()`, `set()`, and `replace()`, all return new URL objects. This enables URLs to be used in sets, as well as dictionary keys.

3.1.3 Query parameters

One of the URL format's most useful features is the mapping formed by the query parameters, sometimes called "query arguments" or "GET parameters". Regardless of what you call them, they are encoded in the query string portion of the URL, and they are very powerful.

In the simplest case, these query parameters can be provided as a dictionary:

```
>>> url = URL.from_text('http://example.com/')
>>> url = url.replace(query={'a': 'b', 'c': 'd'})
>>> url.to_text()
u'http://example.com/?a=b&c=d'
```

Query parameters are actually a type of "multidict", where a given key can have multiple values. This is why the `get()` method returns a list of strings. Keys can also have no value, which is conventionally interpreted as a truthy flag.

```
>>> url = URL.from_text('http://example.com/?a=b&c')
>>> url.get(u'a')
['b']
>>> url.get(u'c')
[None]
>>> url.get('missing')  # returns None
[]
```

Values can be modified and added using `set()` and `add()`.

```
>>> url = url.add(u'x', u'x')
>>> url = url.add(u'x', u'y')
>>> url.to_text()
u'http://example.com/?a=b&c&x=x&x=y'
>>> url = url.set(u'x', u'z')
>>> url.to_text()
u'http://example.com/?a=b&c&x=z'
```

Values can be unset with `remove()`.

```
>>> url = url.remove(u'a')
>>> url = url.remove(u'c')
>>> url.to_text()
u'http://example.com/?x=z'
```

Note how all modifying methods return copies of the URL and do not mutate the URL in place, much like methods on strings.

3.1.4 Origins and backwards-compatibility

Hyperlink's URL is descended directly from twisted.python.url.URL, in all but the literal code-inheritance sense. While a lot of functionality has been incorporated from boltons.urlutils, extra care has been taken to maintain backwards-compatibility for legacy APIs, making Hyperlink's URL a drop-in replacement for Twisted's URL type.

If you are porting a Twisted project to use Hyperlink's URL, and encounter any sort of incompatibility, please do not hesitate to file an issue.

3.2 Hyperlink API

3.2.1 Creation

Before you can work with URLs, you must create URLs. There are two ways to create URLs, from parts and from text.

3.2.2 Transformation

Once a URL is created, some of the most common tasks are to transform it into other URLs and text.

3.2.3 Navigation

Go places with URLs. Simulate browser behavior and perform semantic path operations.

3.2.4 Query Parameters

CRUD operations on the query string multimap.

3.2.5 Attributes

URLs have many parts, and URL objects have many attributes to represent them.

3.2.6 Low-level functions

A couple of notable helpers used by the `URL` type.

3.3 FAQ

There were bound to be questions.

3.3.1 Why not just use text?

URLs were designed as a text format, so, apart from the principle of structuring structured data, why use URL objects?

There are two major advantages of using `URL` over representing URLs as strings. The first is that it's really easy to evaluate a relative hyperlink, for example, when crawling documents, to figure out what is linked:

```
>>> URL.from_text(u'https://example.com/base/uri/').click(u"/absolute")
URL.from_text(u'https://example.com/absolute')
>>> URL.from_text(u'https://example.com/base/uri/').click(u"rel/path")
URL.from_text(u'https://example.com/base/uri/rel/path')
```

The other is that URLs have two normalizations. One representation is suitable for humans to read, because it can represent data from many character sets - this is the Internationalized, or IRI, normalization. The other is the older, US-ASCII-only representation, which is necessary for most contexts where you would need to put a URI. You can convert *between* these representations according to certain rules. `URL` exposes these conversions as methods:

```
>>> URL.from_text(u"https://→example.com/foobar/").to_uri()
URL.from_text(u'https://xn--example-dk9c.com/foo%E2%87%A7bar/')
>>> URL.from_text(u'https://xn--example-dk9c.com/foo%E2%87%A7bar/').to_iri()
URL.from_text(u'https://\\u2192example.com/foo\\u21e7bar/')
```

For more info, see A Tale of Two Representations, above.

3.3.2 How does Hyperlink compare to other libraries?

Hyperlink certainly isn't the first library to provide a Python model for URLs. It just happens to be among the best.

urlparse: Built-in to the standard library (merged into urllib for Python 3). No URL type, requires user to juggle a bunch of strings. Overly simple approach makes it easy to make mistakes.

boltons.urlutils: Shares some underlying implementation. Two key differences. First, the boltons URL is mutable, intended to work like a string factory for URL text. Second, the boltons URL has advanced query parameter mapping type. Complete implementation in a single file.

furl: Not a single URL type, but types for many parts of the URL. Similar approach to boltons for query parameters. Poor netloc handling (support for non-network schemes like mailto). Unlicensed.

purl: Another immutable implementation. Method-heavy API.

rfc3986: Very heavily focused on various types of validation. Large for a URL library, if that matters to you. Exclusively supports URIs, lacking IRI support at the time of writing.

In reality, any of the third-party libraries above do a better job than the standard library, and much of the hastily thrown together code in a corner of a util.py deep in a project. URLs are easy to mess up, make sure you use a tested implementation.

3.3.3 Are URLs really a big deal in 201X?

Hyperlink's first release, in 2017, comes somewhere between 23 and 30 years after URLs were already in use. Is the URL really still that big of a deal?

Look, buddy, I don't know how you got this document, but I'm pretty sure you (and your computer) used one if not many URLs to get here. URLs are only getting more relevant. Buy stock in URLs.

And if you're worried that URLs are just another technology with an obsoletion date planned in advance, I'll direct your attention to the `IPvFuture` rule in the BNF grammar. If it has plans to outlast IPv6, the URL will probably outlast you and me, too.