



Announcing Requests v1.0.0!

2013

3 min read • 674 words

Today marks a major milestone. Requests, Python HTTP for Humans, has finally reached release **v1.0**. This is a big deal.

The transition from version 0.x to 1.0 represents a fundamental shift in software development philosophy. According to semantic versioning principles, this milestone signals the library's evolution from experimental prototype to production-ready infrastructure, establishing a social contract with developers about API stability.

Per [Semantic Versioning](#):

- Major version zero (0.y.z) is for initial development. Anything may change at any time.
- Version 1.0.0 defines the public API.

This release focuses on simplicity, refining of scope, sustainability, and the [future of Python HTTP](#). I re-evaluated every unique feature of Requests, and stripped it down to the bare essentials.

I couldn't be more proud of the result.

Simplicity at its finest

A large number of features have been removed from Requests.

- Redirect resolution is now accomplished with a sexy response generator.

- Built-in OAuth and Kerberos support moved to new [requests](#) org.
- All hooks except `response` removed, which was the only one in serious use.
- Removal of the magic `Response.json` property. Replaced with a method.
- Change of `prefetch` argument to `stream`, to emphasize its purpose.
- Sessions now provide all persistence, connections, and configuration. No magic.
- Session creation no longer has any options. You can modify a session yourself.
- Removal of esoteric ISC license. Replaced with Apache 2.0.
- There are no more global configuration defaults.

| Simplicity is always better than functionality. — Pieter Hintjens

The entire codebase has been rearchitected.

This architectural decision reflects a broader trend in software design toward constraint-based thinking. By removing configuration options, the author embraces the Unix philosophy of "do one thing well" while forcing users into pit-of-success patterns that reduce cognitive overhead and potential misuse.

For example, Requests no longer has **any** configuration.

Previously, there was a configuration mode called `encode_uri`, which gave the user the ability to fully disable Requests' (rather advanced) URI quoting. This helped two or three people get around some server-specific issues.

Now, you can cleanly create your own `Request` instances.

```
>>> req = requests.Request(method='GET', url=..., data=...)
<Request [GET]>
```

And let Requests do all the hard work.

```
>>> prep = req.prepare()  
<PreparedRequest [GET]>  
>>> prep.body  
b'...'
```

Have a problem with the exact value of `url` or `data`? Change them yourself.

```
>>> prep.url = '.../obscure-cornercase.html'  
>>> prep.body = b'exact-bytes-required'
```

Who needs configuration?

Connection Adapters

So, how do we send this new `PreparedRequest` object? Before, you'd call a magical `Request.send` method, which would run about 2000 lines of complex, hard-to-follow, code.

Now, you can send your `PreparedRequest` through a session's connection. Of course, `Requests` comes with a perfect adapter already: `HTTPAdapter`. It seamlessly supports connection pooling, proxies, timeouts, etc. Just like you know and love :)

```
>>> s = requests.Session()  
>>> s.send(prepare())  
<Response [200]>
```

No need for a session? Send it straight through your own connection. Trade them like Pokémon cards!

```
>>> from requests.adapters import HTTPAdapter  
>>> conn = HTTPAdapter()  
>>> conn.send(prepare())  
<Response [200]>
```

A Connection Adapter is a simple class that lets you declare a connection, its persistence, and how to turn a `PreparedRequest` into a `Response`. You can register Connection Adapters in a `Session`, and requests will automatically be sent through the proper adapter.

For example, we could now send requests through a theoretical `WSGIAdapter`:

```
>>> from fakemodule import WSGIAdapter
>>> s = requests.Session()
>>> s.mount('http://staging/', WSGIAdapter())
>>> s.get('http://staging/index.html')
<Response [200]>
```

Beautiful.

Implementing SPDY will be ~60 lines of code, once there's a decent client library. :)

Moving Forward

Request's development will now focus on a few different efforts:

- Bugfixes and improvements, obviously.
- Support for streaming uploads (e.g. `requests.get(...), data=generator()`)
- Support for new protocols (SPDY)
- Communication. Documentation, ecosystem, and community.

It's been a great [two years](#). Thanks to everyone in the community for your continual love and support!

This acknowledgment highlights the symbiotic relationship between open-source maintainers and their communities. The success of libraries like Requests demonstrates how developer experience improvements can create network effects, where ease of use drives adoption, which in turn generates community contributions and feedback loops.

This project would be a fruitless endeavor without you.

Cheers — here's to 1.6 million more downloads.

