sphinxcontrib-httpdomain Documentation

Release 1.7.0

Hong Minhee

Jul 01, 2018

Contents

This contrib extension, *sphinxcontrib.httpdomain*, provides a Sphinx domain for describing HTTP APIs.

See also:

We might support reflection for web framework your webapp depends on. See the following `sphinxcontrib.autohttp` modules:

Module *sphinxcontrib.autohttp.flask* Reflection for Flask webapps.

Module *sphinxcontrib.autohttp.flaskqref* Quick reference rendering with *sphinxcontrib.autohttp.flask*

Module *sphinxcontrib.autohttp.bottle* Reflection for Bottle webapps.

Module *sphinxcontrib.autohttp.tornado* Reflection for Tornado webapps.

In order to use it, add *sphinxcontrib.httpdomain* into `extensions` list of your Sphinx configuration file (`conf.py`):

```
extensions = ['sphinxcontrib.httpdomain']
```

CHAPTER 1

Additional Configuration

http_headers_ignore_prefixes List of HTTP header prefixes which should be ignored in strict mode:

```
http_headers_ignore_prefixes = ['X-']
```

New in version 1.4.0.

Deprecated since version 1.5.0: strict mode no longer warns on non-standard header prefixes.

http_index_ignore_prefixes Strips the leading segments from the endpoint paths by given list of prefixes:

```
http_index_ignore_prefixes = ['/internal', '/_proxy']
```

New in version 1.3.0.

http_index_shortname Short name of the index which will appear on every page:

```
http_index_shortname = 'api'
```

New in version 1.3.0.

http_index_localname Full index name which is used on index page:

```
http_index_localname = "My Project HTTP API"
```

New in version 1.3.0.

http_strict_mode When `True` (default) emits build errors when status codes, methods and headers are looks non-standard:

```
http_strict_mode = True
```

New in version 1.4.0.

CHAPTER 2

Basic usage

There are several provided *directives* that describe HTTP resources.

```
.. http:get:: /users/(int:user_id)/posts/(tag)

   The posts tagged with `tag` that the user (`user_id`) wrote.

   **Example request**:

   .. sourcecode:: http

      GET /users/123/posts/web HTTP/1.1
      Host: example.com
      Accept: application/json, text/javascript

   **Example response**:

   .. sourcecode:: http

      HTTP/1.1 200 OK
      Vary: Accept
      Content-Type: text/javascript

      [
        {
          "post_id": 12345,
          "author_id": 123,
          "tags": ["server", "web"],
          "subject": "I tried Nginx"
        },
        {
          "post_id": 12346,
          "author_id": 123,
          "tags": ["html5", "standards", "web"],
          "subject": "We go to HTML 5"
```

(continues on next page)

(continued from previous page)

```
        }
    ]

:query sort: one of ``hit``, ``created-at``
:query offset: offset number. default is 0
:query limit: limit number. default is 30
:reqheader Accept: the response content type depends on
                   :mailheader:`Accept` header
:reqheader Authorization: optional OAuth token to authenticate
:resheader Content-Type: this depends on :mailheader:`Accept`
                         header of request
:statuscode 200: no error
:statuscode 404: there's no user
```

will be rendered as:

GET /users/(**int:** *user_id*)**/posts/**

tag The posts tagged with *tag* that the user (*user_id*) wrote.

Example request:

```
GET /users/123/posts/web HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

[
  {
    "post_id": 12345,
    "author_id": 123,
    "tags": ["server", "web"],
    "subject": "I tried Nginx"
  },
  {
    "post_id": 12346,
    "author_id": 123,
    "tags": ["html5", "standards", "web"],
    "subject": "We go to HTML 5"
  }
]
```

Query Parameters

- **sort** – one of `hit`, `created-at`
- **offset** – offset number. default is 0
- **limit** – limit number. default is 30

Request Headers

- Accept – the response content type depends on `Accept` header
- Authorization – optional OAuth token to authenticate

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 200 OK – no error
- 404 Not Found – there's no user

Of course, *roles* that refer the directives as well. For example:

```
:http:get:`/users/(int:user_id)/posts/(tag)`
```

will render like:

GET /users/(int:user_id)/posts/(tag)

CHAPTER 3

Directives

.. **http:options::** path
Describes a HTTP resource's OPTIONS method. It can also be referred by *http:options* role.

.. **http:head::** path
Describes a HTTP resource's HEAD method. It can also be referred by *http:head* role.

.. **http:post::** path
Describes a HTTP resource's POST method. It can also be referred by *http:post* role.

.. **http:get::** path
Describes a HTTP resource's GET method. It can also be referred by *http:get* role.

.. **http:put::** path
Describes a HTTP resource's PUT method. It can also be referred by *http:put* role.

.. **http:patch::** path
Describes a HTTP resource's PATCH method. It can also be referred by *http:patch* role.

.. **http:delete::** path
Describes a HTTP resource's DELETE method. It can also be referred by *http:delete* role.

.. **http:trace::** path
Describes a HTTP resource's TRACE method. It can also be referred by *http:trace* role.

.. **http:copy::** path
Describes a HTTP resource's COPY method. It can also be referred by *http:copy* role.

New in version 1.3.0.

.. **http:any::** path
Describes a HTTP resource's which accepts requests with *any* method. Useful for cases when requested resource proxying the request to some other location keeping original request context. It can also be referred by *http:any* role.

New in version 1.3.0.

3.1 Options

New in version 1.3.0.

Additionally, you may specify custom options to the directives:

noindex Excludes specific directive from HTTP routing table.

```
.. http:get:: /users/(int:user_id)/posts/(tag)
   :noindex:
```

deprecated Marks the method as deprecated in HTTP routing table.

```
.. http:get:: /users/(int:user_id)/tagged_posts
   :deprecated:
```

synopsis Adds short description for HTTP routing table.

```
.. http:get:: /users/(int:user_id)/posts/(tag)
   :synopsis: Returns posts by the specified tag for the user
```

CHAPTER 4

Resource Fields

Inside HTTP resource description directives like `get`, reStructuredText field lists with these fields are recognized and formatted nicely:

param, parameter, arg, argument Description of URL parameter.

queryparameter, queryparam, qparam, query Description of parameter passed by request query string.

It optionally can be typed, all the query parameters will have obviously string types though. But it's useful if there's conventions for it.

Changed in version 1.1.9: It can be typed e.g.:

```
:query string title: the post title
:query string body: the post body
:query boolean sticky: whether it's sticky or not
```

formparameter, formparam, fparam, form Description of parameter passed by request content body, encoded in *application/x-www-form-urlencoded* or *multipart/form-data*.

jsonparameter, jsonparam, json Description of a parameter passed by request content body, encoded in *application/json*.

Deprecated since version 1.3.0: Use `reqjsonobj`/`reqjson`/`<jsonobj`/`<json` and `reqjsonarr`/`<jsonarr` instead.

New in version 1.1.8.

Changed in version 1.1.9: It can be typed e.g.:

```
:jsonparam string title: the post title
:jsonparam string body: the post body
:jsonparam boolean sticky: whether it's sticky or not
```

reqjsonobj, reqjson, <jsonobj, <json Description of a single field of JSON object passed by request body, encoded in *application/json*. The key difference from `json` is explicitly defined use-case (request/response) of the described object.

```
:<json string title: the post title
:<json string body: the post body
:<json boolean sticky: whether it's sticky or not
```

New in version 1.3.0.

resjsonobj, resjson, >jsonobj, >json Description of a single field of JSON object returned with response body, encoded in *application/json*.

```
:>json boolean ok: Operation status
```

New in version 1.3.0.

reqjsonarr, <jsonarr resjsonarr, >jsonarr

Similar to <json and >json respectively, but uses for describing objects schema inside of returned array.

Let's say, the response contains the following data:

```
[{"id": "foo", "ok": true}, {"id": "bar", "error": "forbidden", "reason":
↪"sorry"}]
```

Then we can describe it in the following way:

```
:>jsonarr boolean ok: Operation status. Not present in case of error
:>jsonarr string id: Object ID
:>jsonarr string error: Error type
:>jsonarr string reason: Error reason
```

New in version 1.3.0.

```
:>json boolean status: Operation status
```

requestheader, reqheader, >header Description of request header field.

New in version 1.1.9.

responseheader, resheader, <header Description of response header field.

New in version 1.1.9.

statuscode, status, code Description of response status code.

For example:

```
.. http:post:: /posts/(int:post_id)

   Replies a comment to the post.

   :param post_id: post's unique id
   :type post_id: int
   :form email: author email address
   :form body: comment body
   :reqheader Accept: the response content type depends on
                      :mailheader:`Accept` header
   :reqheader Authorization: optional OAuth token to authenticate
   :resheader Content-Type: this depends on :mailheader:`Accept`
                            header of request
   :status 302: and then redirects to :http:get:`/posts/(int:post_id)`
```

(continues on next page)

(continued from previous page)

```
   :status 400: when form parameters are missing

.. http:get:: /posts/(int:post_id)

   Fetches the post

   (...)
```

It will render like this:

POST /posts/(**int:** *post_id*)
Replies a comment to the post.

Parameters

- **post_id** (*int*) – post's unique id

Form Parameters

- **email** – author email address
- **body** – comment body

Request Headers

- Accept – the response content type depends on *Accept* header
- Authorization – optional OAuth token to authenticate

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 302 Found – and then redirects to *GET /posts/(int:post_id)*
- 400 Bad Request – when form parameters are missing

GET /posts/(**int:** *post_id*)
Fetches the post

(...)

CHAPTER 5

Roles

:http:options:
Refers to the *http:options* directive.

:http:head:
Refers to the *http:head* directive.

:http:post:
Refers to the *http:post* directive.

:http:get:
Refers to the *http:get* directive.

:http:put:
Refers to the *http:put* directive.

:http:patch:
Refers to the *http:patch* directive.

:http:delete:
Refers to the *http:delete* directive.

:http:trace:
Refers to the *http:trace* directive.

:http:copy:
Refers to the *http:copy* directive.

:http:any:
Refers to the *http:any* directive.

:http:statuscode:
A reference to a HTTP status code. The text "*code Status Name*" is generated; in the HTML output, this text is a hyperlink to a web reference of the specified status code.

For example:

```
- :http:statuscode:`404`
- :http:statuscode:`200 OK`
```

will be rendered as:

- 404 Not Found
- 200 OK

Changed in version 1.3.0: It becomes to provide references to specification sections.

:http:method:
A reference to a HTTP method (also known as *verb*). In the HTML output, this text is a hyperlink to a web reference of the specified HTTP method.

For example:

```
It accepts :http:method:`post` only.
```

It will render like this:

> It accepts POST only.

:mimetype:
Exactly it doesn't belong to HTTP domain, but standard domain. It refers to the MIME type like *text/html*.

:mailheader:
Deprecated since version 1.3.0: Use *http:header* instead.

:http:header:
Similar to *mimetype* role, it doesn't belong to HTTP domain, but standard domain. It refers to the HTTP request/response header field like Content-Type.

If the HTTP header is known, the text is a hyperlink to a web reference of the specified header.

Known HTTP headers:

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Accept-Ranges
- Age
- Allow
- Authorization
- Cache-Control
- Connection
- Content-Encoding
- Content-Language
- Content-Length
- Content-Location
- Content-MD5
- Content-Range
- Content-Type
- Cookie

- Date
- Destination
- ETag
- Expect
- Expires
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Last-Modified
- Last-Event-ID
- Link
- Location
- Max-Forwards
- Pragma
- Proxy-Authenticate
- Proxy-Authorization
- Range
- Referer
- Retry-After
- Server
- Set-Cookie
- TE
- Trailer
- Transfer-Encoding
- Upgrade
- User-Agent
- Vary
- Via
- WWW-Authenticate
- Warning

New in version 1.3.0.

Changed in version 1.5.0: No longer emits warnings for unrecognized headers

CHAPTER 6

sphinxcontrib.autohttp.flask — Exporting API reference from Flask app

New in version 1.1.

It generates RESTful HTTP API reference documentation from a Flask application's routing table, similar to sphinx.ext.autodoc.

In order to use it, add *sphinxcontrib.autohttp.flask* into extensions list of your Sphinx configuration (conf.py) file:

```
extensions = ['sphinxcontrib.autohttp.flask']
```

For example:

```
.. autoflask:: autoflask_sampleapp:app
   :undoc-static:
```

will be rendered as:

GET /
Home page.

GET / (*user*) **/posts/**
int: *post_id* User's post.

Parameters

- **user** – user login name
- **post_id** – post unique id

Status Codes

- 200 OK – when user and post exists
- 404 Not Found – when user and post doesn't exist

GET / (*user*)
User profile page.

Parameters

- **user** – user login name

Status Codes

- 200 OK – when user exists
- 404 Not Found – when user doesn't exist

`.. autoflask:: module:app`

New in version 1.1.

Generates HTTP API references from a Flask application. It takes an import name, like:

```
your.webapplication.module:app
```

Colon character (`:`) separates module path and application variable. Latter part can be more complex:

```
your.webapplication.module:create_app(config='default.cfg')
```

It's useful when a Flask application is created from the factory function (`create_app()`, in the above example).

It takes several flag options as well.

endpoints Endpoints to generate a reference.

```
.. autoflask:: yourwebapp:app
   :endpoints: user, post, friends
```

will document `user()`, `post()`, and `friends()` view functions, and

```
.. autoflask:: yourwebapp:app
   :endpoints:
```

will document all endpoints in the flask app.

For compatibility, omitting this option will produce the same effect like above.

New in version 1.1.8.

undoc-endpoints Excludes specified endpoints from generated references.

For example:

```
.. autoflask:: yourwebapp:app
   :undoc-endpoints: admin, admin_login
```

will exclude `admin()`, `admin_login()` view functions.

Note: It is worth noting that the names of endpoints that are applied to blueprints are prefixed with the blueprint's name (e.g. blueprint.endpoint).

Note: While the undoc-members flag of `sphinx.ext.autodoc` extension includes members without docstrings, `undoc-endpoints` option has nothing to do with docstrings. It just excludes specified endpoints.

---

blueprints Only include specified blueprints in generated references.

New in version 1.1.9.

undoc-blueprints Excludes specified blueprints from generated references.

New in version 1.1.8.

modules Only include specified view modules in generated references.

For example:

```
.. autoflask:: yourwebapp:app
   :modules: yourwebapp.views.admin
```

will include only views in `yourwebapp.views.admin` module

New in version 1.5.0.

undoc-modules Excludes specified view modules from generated references.

New in version 1.5.0.

undoc-static Excludes a view function that serves static files, which is included in Flask by default.

order Determines the order in which endpoints are listed. Currently only `path` is supported.

For example:

```
.. autoflask:: yourwebapp:app
   :endpoints:
   :order: path
```

will document all endpoints in the flask app, ordered by their route paths.

New in version 1.5.0.

groupby Determines the strategy to group paths. Currently only `view` is supported. Specified this will group paths by their view functions.

New in version 1.6.0.

include-empty-docstring View functions that don't have docstring (`__doc__`) are excluded by default. If this flag option has given, they become included also.

New in version 1.1.2.

CHAPTER 7

sphinxcontrib.autohttp.flaskqref — Quick API reference for Flask app

New in version 1.5.0.

This generates a quick API reference table for the route documentation produced by *sphinxcontrib.autohttp.flask*

To use it, both *sphinxcontrib.autohttp.flask* and *sphinxcontrib.autohttp.flaskqref* need to be added into the extensions of your configuration (`conf.py`) file:

```
extensions = ['sphinxcontrib.autohttp.flask',
              'sphinxcontrib.autohttp.flaskqref']
```

.. qrefflask:: module:app
New in version 1.5.0.

Generates HTTP API references from a Flask application and places these in a list-table with quick reference links. The usage and options are identical to that of *sphinxcontrib.autohttp.flask*

7.1 Basic usage

You typically would place the quick reference table near the top of your docco and use .. *autoflask::* further down.

Routes that are to be included in the quick reference table require the following rst comment to be added to their doc string:

```
.. :quickref: [<resource>;] <short description>
```

<resource> is optional, if used a semi-colon separates it from <short description> The table is grouped and sorted by <resource>.

<resource> This is the resource name of the operation. The name maybe the same for a number of operations and enables grouping these together.

<short description> A brief description what the operation does.

For example:

```
@app.route('/<user>')
def user(user):
    """User profile page.

    .. :quickref: User; Get Profile Page

    my docco here
    """
    return 'hi, ' + user
```

The quick reference table is defined as:

```
.. qrefflask:: autoflask_sampleapp:app
   :undoc-static:
```

Using the autoflask_sampleapp with .. *:quickref:* annotations, this is rendered as:

Resource	Operation	Description
Home	*GET /*	The Home page
User	*GET /(user)/posts/(int:post_id)*	Save user id
	GET /(user)	Get Profile Page

CHAPTER 8

sphinxcontrib.autohttp.bottle — Exporting API reference from Bottle app

It generates RESTful HTTP API reference documentation from a Bottle application's routing table, similar to `sphinx.ext.autodoc`.

In order to use it, add *sphinxcontrib.autohttp.bottle* into `extensions` list of your Sphinx configuration (`conf.py`) file:

```
extensions = ['sphinxcontrib.autohttp.bottle']
```

For example:

```
.. autobottle:: autobottle_sampleapp:app
```

will be rendered as:

GET /
Home page.

GET / (*user*)
User profile page.

Parameters

- **user** – user login name

Status Codes

- 200 OK – when user exists
- 404 Not Found – when user doesn't exist

GET / (*user*) **/posts/**
post_id: *int* User's post.

Parameters

- **user** – user login name

- **post_id** – post unique id

Status Codes

- 200 OK – when user and post exists
- 404 Not Found – when user and post doesn't exist

`.. autobottle:: module:app`

Generates HTTP API references from a Bottle application. It takes an import name, like:

```
your.webapplication.module:app
```

Colon character (`:`) separates module path and application variable. Latter part can be more complex:

```
your.webapplication.module:create_app(config='default.cfg')
```

It's useful when a Bottle application is created from the factory function (`create_app()`, in the above example).

It takes several flag options as well.

endpoints Endpoints to generate a reference.

```
.. autobottle:: yourwebapp:app
   :endpoints: user, post, friends
```

will document `user()`, `post()`, and `friends()` view functions, and

```
.. autobottle:: yourwebapp:app
   :endpoints:
```

will document all endpoints in the bottle app.

For compatibility, omitting this option will produce the same effect like above.

undoc-endpoints Excludes specified endpoints from generated references.

For example:

```
.. autobottle:: yourwebapp:app
   :undoc-endpoints: admin, admin_login
```

will exclude `admin()`, `admin_login()` view functions.

Note: While the undoc-members flag of `sphinx.ext.autodoc` extension includes members without docstrings, `undoc-endpoints` option has nothing to do with docstrings. It just excludes specified endpoints.

---

include-empty-docstring View functions that don't have docstring (`__doc__`) are excluded by default. If this flag option has given, they become included also.

CHAPTER 9

sphinxcontrib.autohttp.tornado — Exporting API reference from Tornado app

It generates RESTful HTTP API reference documentation from a Tornado application's routing table, similar to `sphinx.ext.autodoc`.

In order to use it, add *`sphinxcontrib.autohttp.tornado`* into `extensions` list of your Sphinx configuration (`conf.py`) file:

```
extensions = ['sphinxcontrib.autohttp.tornado']
```

For example:

```
.. autotornado:: autotornado_sampleapp:app
```

will be rendered as:

GET /
Home page.

GET /(?P<user>[a-z0-9]+)
User profile page.

Parameters

- **user** – user login name

Status Codes

- 200 OK – when user exists
- 404 Not Found – when user doesn't exist

GET /(?P<user>[a-z0-9]+)/posts/(?P<post_id>[d+]+)
User's post.

Parameters

- **user** – user login name

- **post_id** – post unique id

Status Codes

- 200 OK – when user and post exists
- 404 Not Found – when user and post doesn't exist

`.. autotornado:: module:app`

Generates HTTP API references from a Tornado application. It takes an import name, like:

```
your.webapplication.module:app
```

Colon character (`:`) separates module path and application variable.

It takes several flag options as well.

endpoints Endpoints to generate a reference.

```
.. autotornado:: yourwebapp:app
   :endpoints: User.get, User.post, Friends.get
```

will document the `get()` and `post()` methods of the `User RequestHandler` and the `get()` method of the `Friend RequestHandler`, while

```
.. autotornado:: yourwebapp:app
   :endpoints:
```

will document all endpoints in the tornado app.

For compatibility, omitting this option will produce the same effect like above.

undoc-endpoints Excludes specified endpoints from generated references.

For example:

```
.. autotornado:: yourwebapp:app
   :undoc-endpoints: admin, admin_login
```

will exclude `admin()`, `admin_login()` view functions.

Note: While the undoc-members flag of `sphinx.ext.autodoc` extension includes members without docstrings, `undoc-endpoints` option has nothing to do with docstrings. It just excludes specified endpoints.

---

include-empty-docstring View functions that don't have docstring (`__doc__`) are excluded by default. If this flag option has given, they become included also.

CHAPTER 10

Author and License

The *sphinxcontrib.httpdomain* and sphinxcontrib.autohttp, parts of sphinxcontrib, are written by Hong Minhee and distributed under BSD license.

The source code is maintained under the sphinx-contrib project in the httpdomain repository

```
$ git clone https://github.com/sphinx-contrib/httpdomain.git
$ cd httpdomain
```

orphan

CHAPTER 11

Changelog

11.1 Version 1.7.0

Released on July 1, 2018.

- Implemented `:autoquickref:` option that use available informations to build a `quickref`. [pull request #9 by Alexandre Bonnetain]
- Improved *sphinxcontrib.autohttp.tornado* compatibility with Tornado 4.5 and newer. *Tornado 4.5 <http://www.tornadoweb.org/en/stable/releases/v4.5.0.html>* introduced the `Rule` class and made `URLSpec` a subclass of it, so certain rule attributes required updating. [issue #7, pull request #11 by Robert Zeigler]

11.2 Version 1.6.1

Released on March 3, 2018.

- Remove references to the `sphinx.util.compat` module which was deprecated in Sphinx 1.6 and removed in 1.7. [issue #5, pull request #4 by Jeremy Cline]
- Made *sphinxcontrib.autohttp.tornado* compatible with Tornado 4.5 and newer. *Tornado 4.5 <http://www.tornadoweb.org/en/stable/releases/v4.5.0.html>* removed the `handlers` attribute from `tornado.web.Application`. [pull request #3 by Dave Shawley]

11.3 Version 1.6.0

Released on January 13, 2018.

- Minimum compatible version of Sphinx became changed to 1.5.
- Fixed a bug that prevented building `sphinxcontrib.autohttp` from building properly with Sphinx 1.6 or higher. [old issue #182, old pull request #152 by Dave Shawley]

- Use HTTPS for `:rfc:` generated links. [old pull request #144 by Devin Sevilla]
- Added `groupby` option to *autoflask* directive. It makes paths be grouped by their view functions. [old pull request #147 by Jiangge Zhang]
- Fixed a bug that *autoflask* directive had excluded nonsignificant routes with HEAD/OPTIONS. [old issue #165]

11.4 Version 1.5.0

Released on May 30, 2016.

- Added *sphinxcontrib.autohttp.flaskqref* for generating quick reference table. [old pull request #80, old pull request #100 by Harry Raaymakers]
- *autoflask* now supports `:modules:` and `:undoc-modules:` arguments, used to filter documented flask endpoints by view module [old pull request #102 by Ivelin Slavov]
- Added `:order:` option to *autoflask* directive. [old pull request #103 by Justin Gruca]
- HTTP message headers become to link the recent RFCs (**RFC 7230**, **RFC 7231**, **RFC 7232**, **RFC 7233**, **RFC 7234**, **RFC 7235**, **RFC 7236**, **RFC 7237**, that are separated to multiple RFCs from the old one) instead of **RFC 2615** which is replaced by them in 2014. [old pull request #105, old pull request #106 by Alex C. (iscandr)]
- Support `resolve_any_xref` method introduced since Sphinx 1.3 [old pull request #108 by Takayuki Shimizukawa]
- It no more warns non-standard message headers without `X-` prefix according as the deprecation of the practice of prefixing the names of unstandardized parameters with `X-` in all IETF protocols since June 2012 by **RFC 6648**. [old pull request #114 by Dolan Murvihill]
- Fixed performance bottleneck in doctree lookup by adding a cache for it. [old pull request #115 by Kai Lautaportti]
- Added 451 Unavailable For Legal Reasons to *http:statuscode*. [old pull request #117 by Xavier Oliver]

11.5 Version 1.4.0

Released on August 13, 2015.

- Added 429 Too Many Requests as a valid *http:statuscode*. [old pull request #81 by DDBReloaded]
- Became to not resolve references if they can't be resolved. [old pull request #87 by Ken Robbins]
- Became to preserve endpoint ordering when `:endpoints:` option is given. [old pull request #88 by Dan Callaghan]
- Added status codes for WebDAV. [old pull request #92 by Ewen Cheslack-Postava]
- Added CORS headers. [old pull request #96 by Tomi Pieviläinen]
- Now *sphinxcontrib.autohttp.flask* supports multiple paths for endpoints using same HTTP method. [old pull request #97 by Christian Felder]

11.6 Version 1.3.0

Released on July 31, 2014.

- `jsonparameter`/`jsonparam`/`json` became deprecated and split into `reqjsonobj`/`reqjson`/`<jsonobj`/`<json` and `reqjsonarr`/`<jsonarr`. [old issue #55, old pull request #72 by Alexander Shorin]
- Support synopsis (short description in HTTP index), deprecation and noindex options for resources. [old issue #55, old pull request #72 by Alexander Shorin]
- Stabilize order of index items. [old issue #55, old pull request #72 by Alexander Shorin]
- Added *`http:any`* directive and *`http:any`* role for `ANY` method. [old issue #55, old pull request #72 by Alexander Shorin]
- Added *`http:copy`* directive and *`http:copy`* role for `COPY` method. [old issue #55, old pull request #72 by Alexander Shorin]
- Added *`http:header`* role that also creates reference to the related specification. [old issue #55, old pull request #72 by Alexander Shorin]
- *`http:statuscode`* role became to provide references to specification sections. [old issue #55, old pull request #72 by Alexander Shorin]
- Fixed Python 3 incompatibility of `autohttp.tornado`. [old pull request #61 by Dave Shawley]

11.7 Version 1.2.1

Released on March 31, 2014.

- Fixed broken Python 2.6 compatibility. [old pull request #41 by Kien Pham]
- Added missing link to six dependency.

11.8 Version 1.2.0

Released on October 19, 2013.

- Python 3 support! [old pull request #34 by murchik, old pull request #39 Donald Stufft]
- Added support for Tornado webapps. (*`sphinxcontrib.autohttp.tornado`*) [old pull request #38 by Rodrigo Machado]

11.9 Version 1.1.9

Released on August 8, 2013.

- Now Bottle apps can be loaded by `autohttp`. See *`sphinxcontrib.autohttp.bottle`* module. [patch by Jameel Al-Aziz]
- Added `:reqheader:` and `:resheader:` option flags.
- `:jsonparameter:` can be typed. [old pull request #31 by Chuck Harmston]
- `:queryparameter:` can be typed. [old pull request #37 by Viktor Haag]
- *`autoflask`* and *`autobottle`* directives now allow empty `:endpoints:`, `:undoc-endpoints:`, and `:blueprints:` arguments. [old pull request #33 by Michael Twomey]

11.10 Version 1.1.8

Released on April 10, 2013.

- Added better support for docstrings in `flask.views.MethodView`. [old pull request #26 by Simon Metson]
- Added `:jsonparameter:` along side `:form:` and `:query:` flag options. [old pull request #25 by Adam Lowry]
- Fixed issue with undefined `Value` and `umethod` variables. [old pull request #23 by Sebastian Kalinowski and old pull request #24 by Viktor Haag]
- Now `http` Pygments lexer can Handle continuous header lines well.
- Added `:undoc-blueprints:` flag option to *`autoflask`* directive. [old pull request #21 by Roman Podolyaka]
- Fixed old issue #29, a bug that *`autoflask`* directive raised `UnicodeDecodeError` when it contains non-ASCII characters. [old issue #29 and old pull request #18 by Eunchong Yu]
- Added `:endpoints:` flag option to *`autoflask`* directive. [old pull request #17 by Eunchong Yu]

11.11 Version 1.1.7

Released on March 28, 2012.

- Added PATCH method support. See *`http:patch`* role and *`http:patch`* directive. [old pull request #9 and old pull request #10 by Jeffrey Finkelstein]
- The HTTP routing table can be grouped based on prefix by specifying `http_index_ignore_prefixes` config in list of common prefixes to ignore. [old pull request #7 and old pull request #8 by Andrey Popp]
- The order of HTTP routing table now provides sorting by path as key. Previously it was sorted by HTTP method and then by path, which is non-intuitive. [old pull request #7 and old pull request #8 by Andrey Popp]

11.12 Version 1.1.6

Released on December 16, 2011.

- Added `http` custom lexer for Pygments so that HTTP sessions can be highlighted in `code-block` or `sourcecode` directives.

11.13 Version 1.1.5

Released on July 6, 2011.

- Flask 0.6–0.7 compatibility. Flask renamed `static_path` attribute to `static_url_path`, so *`autoflask`* also reflect the change. [old pull request #1 by Jeffrey Finkelstein]

11.14 Version 1.1.4

Released on June 8, 2011.

- CPython compatibility
- PyPy compatibility

11.15 Version 1.1.3

Released on June 8, 2011.

- PyPy compatibility

11.16 Version 1.1.2

Released on June 4, 2011.

- Added `:include-empty-docstring:` flag option.

11.17 Version 1.1.1

Released on June 4, 2011.

- Fixed a backward incompatibility bug.

11.18 Version 1.1

Released on June 4, 2011.

- Added *autoflask* directive.

11.19 Version 1.0

Released on June 2, 2011. The first release.

Python Module Index

S

HTTP Routing Table

/

```
GET /, 27
```

/(?P<user>[a-z0-9]+)

```
GET /(?P<user>[a-z0-9]+), 27
GET /(?P<user>[a-z0-9]+)/posts/(?P<post_id>[d+]+),
      27
```

/(user)

```
GET /(user), 25
GET /(user)/posts/(int:post_id), 19
GET /(user)/posts/(post_id:int), 25
```

/posts

```
GET /posts/(int:post_id), 13
POST /posts/(int:post_id), 13
```

/users

```
GET /users/(int:user_id)/posts/(tag), 6
```

Index