rfc3986 Documentation

Release 1.0.0

Ian Stapleton Cordasco

Apr 08, 2020

Contents:

`rfc3986` is a Python implementation of **RFC 3986** including validation and authority parsing. This module also supports **RFC 6874** which adds support for zone identifiers to IPv6 Addresses.

The maintainers strongly suggest using pip to install `rfc3986`. For example,

```
$ pip install rfc3986
$ python -m pip install rfc3986
$ python3.6 -m pip install rfc3986
```

CHAPTER 1

User Documentation

`rfc3986` has several API features and convenience methods. The core of `rfc3986`'s API revolves around parsing, validating, and building URIs.

There is an API to provide compatibility with `urllib.parse`, there is an API to parse a URI as a URI Reference, there's an API to provide validation of URIs, and finally there's an API to build URIs.

Note: There's presently no support for IRIs as defined in **RFC 3987**.

`rfc3986` parses URIs much differently from `urllib.parse` so users may see some subtle differences with very specific URLs that contain rough edgecases. Regardless, we do our best to implement the same API so you should be able to seemlessly swap `rfc3986` for `urlparse`.

1.1 Parsing a URI

There are two ways to parse a URI with `rfc3986`

1. *`rfc3986.api.uri_reference()`*

   This is best when you're **not** replacing existing usage of `urllib.parse`. This also provides convenience methods around safely normalizing URIs passed into it.

2. *`rfc3986.api.urlparse()`*

   This is best suited to completely replace `urllib.parse.urlparse()`. It returns a class that should be indistinguishable from `urllib.parse.ParseResult`

Let's look at some code samples.

1.1.1 Some Examples

First we'll parse the URL that points to the repository for this project.

```
url = rfc3986.urlparse('https://github.com/sigmavirus24/rfc3986')
```

Then we'll replace parts of that URL with new values:

```
print(url.copy_with(
    userinfo='username:password',
    port='443',
).unsplit())
```

```
https://username:password@github.com:443/sigmavirus24/rfc3986
```

This, however, does not change the current `url` instance of `ParseResult`. As the method name might suggest, we're copying that instance and then overriding certain attributes. In fact, we can make as many copies as we like and nothing will change.

```
print(url.copy_with(
    scheme='ssh',
    userinfo='git',
).unsplit())
```

```
ssh://git@github.com/sigmavirus24/rfc3986
```

```
print(url.scheme)
```

```
https
```

We can do similar things with URI References as well.

```
uri = rfc3986.uri_reference('https://github.com/sigmavirus24/rfc3986')
```

```
print(uri.copy_with(
    authority='username:password@github.com:443',
    path='/sigmavirus24/github3.py',
).unsplit())
```

```
https://username:password@github.com:443/sigmavirus24/github3.py
```

However, URI References may have some unexpected behaviour based strictly on the RFC.

Finally, if you want to remove a component from a URI, you may pass `None` to remove it, for example:

```
print(uri.copy_with(path=None).unsplit())
```

```
https://github.com
```

This will work on both URI References and Parse Results.

1.1.2 And Now For Something Slightly Unusual

If you are familiar with GitHub, GitLab, or a similar service, you may have interacted with the "SSH URL" for some projects. For this project, the SSH URL is:

```
git@github.com:sigmavirus24/rfc3986
```

Let's see what happens when we parse this.

```
>>> rfc3986.uri_reference('git@github.com:sigmavirus24/rfc3986')
URIReference(scheme=None, authority=None,
path=u'git@github.com:sigmavirus24/rfc3986', query=None, fragment=None)
```

There's no scheme present, but it is apparent to our (human) eyes that `git@github.com` should not be part of the path. This is one of the areas where `rfc3986` suffers slightly due to its strict conformance to **RFC 3986**. In the RFC, an authority must be preceded by `//`. Let's see what happens when we add that to our URI

```
>>> rfc3986.uri_reference('//git@github.com:sigmavirus24/rfc3986')
URIReference(scheme=None, authority=u'git@github.com:sigmavirus24',
path=u'/rfc3986', query=None, fragment=None)
```

Somewhat better, but not much.

Note: The maintainers of `rfc3986` are working to discern better ways to parse these less common URIs in a reasonable and sensible way without losing conformance to the RFC.

1.2 Validating URIs

While not as difficult as validating an email address, validating URIs is tricky. Different parts of the URI allow different characters. Those sets sometimes overlap and othertimes they don't and it's not very convenient. Luckily, `rfc3986` makes validating URIs far simpler.

1.2.1 Example Usage

First we need to create an instance of a *`Validator`* which takes no parameters. After that we can call methods on the instance to indicate what we want to validate.

Allowing Only Trusted Domains and Schemes

Let's assume that we're building something that takes user input for a URL and we want to ensure that URL is only ever using a specific domain with https. In that case, our code would look like this:

```
>>> from rfc3986 import validators, uri_reference
>>> user_url = 'https://github.com/sigmavirus24/rfc3986'
>>> validator = validators.Validator().allow_schemes(
...     'https',
... ).allow_hosts(
...     'github.com',
... )
>>> validator.validate(uri_reference(
...     'https://github.com/sigmavirus24/rfc3986'
... ))
>>> validator.validate(uri_reference(
...     'https://github.com/'
... ))
```

(continues on next page)

(continued from previous page)

```
>>> validator.validate(uri_reference(
...     'http://example.com'
... ))
Traceback (most recent call last):
...
rfc3986.exceptions.UnpermittedComponentError
```

First notice that we can easily reuse our validator object for each URL. This allows users to not have to constantly reconstruct Validators for each bit of user input. Next, we have three different URLs that we validate:

1. `https://github.com/sigmavirus24/rfc3986`
2. `https://github.com/`
3. `http://example.com`

As it stands, our validator will allow the first two URLs to pass but will fail the third. This is specifically because we only allow URLs using `https` as a scheme and `github.com` as the domain name.

Preventing Leaks of User Credentials

Next, let's imagine that we want to prevent leaking user credentials. In that case, we want to ensure that there is no password in the user information portion of the authority. In that case, our new validator would look like this:

```
>>> from rfc3986 import validators, uri_reference
>>> user_url = 'https://github.com/sigmavirus24/rfc3986'
>>> validator = validators.Validator().allow_schemes(
...     'https',
... ).allow_hosts(
...     'github.com',
... ).forbid_use_of_password()
>>> validator.validate(uri_reference(
...     'https://github.com/sigmavirus24/rfc3986'
... ))
>>> validator.validate(uri_reference(
...     'https://github.com/'
... ))
>>> validator.validate(uri_reference(
...     'http://example.com'
... ))
Traceback (most recent call last):
...
rfc3986.exceptions.UnpermittedComponentError
>>> validator.validate(uri_reference(
...     'https://sigmavirus24@github.com'
... ))
>>> validator.validate(uri_reference(
...     'https://sigmavirus24:not-my-real-password@github.com'
... ))
Traceback (most recent call last):
...
rfc3986.exceptions.PasswordForbidden
```

Requiring the Presence of Components

Up until now, we have assumed that we will get a URL that has the appropriate components for validation. For example, we assume that we will have a URL that has a scheme and hostname. However, our current validation doesn't require those items exist.

```
>>> from rfc3986 import validators, uri_reference
>>> user_url = 'https://github.com/sigmavirus24/rfc3986'
>>> validator = validators.Validator().allow_schemes(
...     'https',
... ).allow_hosts(
...     'github.com',
... ).forbid_use_of_password()
>>> validator.validate(uri_reference('//github.com'))
>>> validator.validate(uri_reference('https:/'))
```

In the first case, we have a host name but no scheme and in the second we have a scheme and a path but no host. If we want to ensure that those components are there and that they are *always* what we allow, then we must add one last item to our validator:

```
>>> from rfc3986 import validators, uri_reference
>>> user_url = 'https://github.com/sigmavirus24/rfc3986'
>>> validator = validators.Validator().allow_schemes(
...     'https',
... ).allow_hosts(
...     'github.com',
... ).forbid_use_of_password(
... ).require_presence_of(
...     'scheme', 'host',
... )
>>> validator.validate(uri_reference('//github.com'))
Traceback (most recent call last):
...
rfc3986.exceptions.MissingComponentError
>>> validator.validate(uri_reference('https:/'))
Traceback (most recent call last):
...
rfc3986.exceptions.MissingComponentError
>>> validator.validate(uri_reference('https://github.com'))
>>> validator.validate(uri_reference(
...     'https://github.com/sigmavirus24/rfc3986'
... ))
```

Checking the Validity of Components

As of version 1.1.0, `rfc3986` allows users to check the validity of a URI Reference using a *`Validator`*. Along with the above examples we can also check that a URI is valid per **RFC 3986**. The validation of the components is pre-determined so all we need to do is specify which components we want to validate:

```
>>> from rfc3986 import validators, uri_reference
>>> valid_uri = uri_reference('https://github.com/')
>>> validator = validators.Validator().allow_schemes(
...     'https',
... ).allow_hosts(
...     'github.com',
... ).forbid_use_of_password(
```

(continues on next page)

(continued from previous page)

```
... ).require_presence_of(
...     'scheme', 'host',
... ).check_validity_of(
...     'scheme', 'host', 'path',
... )
>>> validator.validate(valid_uri)
>>> invalid_uri = valid_uri.copy_with(path='/#invalid/path')
>>> validator.validate(invalid_uri)
Traceback (most recent call last):
...
rfc3986.exceptions.InvalidComponentsError
```

Paths are not allowed to contain a # character unless it's percent-encoded. This is why our `invalid_uri` raises an exception when we attempt to validate it.

1.3 Building URIs

Constructing URLs often seems simple. There are some problems with concatenating strings to build a URL:

- Certain parts of the URL disallow certain characters
- Formatting some parts of the URL is tricky and doing it manually isn't fun

To make the experience better `rfc3986` provides the *`URIBuilder`* class to generate valid *`URIReference`* instances. The *`URIBuilder`* class will handle ensuring that each component is normalized and safe for real world use.

1.3.1 Example Usage

Note: All of the methods on a *`URIBuilder`* are chainable (except *`finalize()`*).

Let's build a basic URL with just a scheme and host. First we create an instance of *`URIBuilder`*. Then we call *`add_scheme()`* and *`add_host()`* with the scheme and host we want to include in the URL. Then we convert our builder object into a *`URIReference`* and call *`unsplit()`*.

```
>>> from rfc3986 import builder
>>> print(builder.URIBuilder().add_scheme(
...     'https'
... ).add_host(
...     'github.com'
... ).finalize().unsplit())
https://github.com
```

It is possible to update an existing URI by constructing a builder from an instance of *`URIReference`* or a textual representation:

```
>>> from rfc3986 import builder
>>> print(builder.URIBuilder.from_uri("http://github.com").add_scheme(
...     'https'
... ).finalize().unsplit())
https://github.com
```

Each time you invoke a method, you get a new instance of a *URIBuilder* class so you can build several different URLs from one base instance.

```
>>> from rfc3986 import builder
>>> github_builder = builder.URIBuilder().add_scheme(
...     'https'
... ).add_host(
...     'api.github.com'
... )
>>> print(github_builder.add_path(
...     '/users/sigmavirus24'
... ).finalize().unsplit())
https://api.github.com/users/sigmavirus24
>>> print(github_builder.add_path(
...     '/repos/sigmavirus24/rfc3986'
... ).finalize().unsplit())
https://api.github.com/repos/sigmavirus24/rfc3986
```

`rfc3986` makes adding authentication credentials convenient. It takes care of making the credentials URL safe. There are some characters someone might want to include in a URL that are not safe for the authority component of a URL.

```
>>> from rfc3986 import builder
>>> print(builder.URIBuilder().add_scheme(
...     'https'
... ).add_host(
...     'api.github.com'
... ).add_credentials(
...     username='us3r',
...     password='p@ssw0rd',
... ).finalize().unsplit())
https://us3r:p%40ssw0rd@api.github.com
```

Further, `rfc3986` attempts to simplify the process of adding query parameters to a URL. For example, if we were using Elasticsearch, we might do something like:

```
>>> from rfc3986 import builder
>>> print(builder.URIBuilder().add_scheme(
...     'https'
... ).add_host(
...     'search.example.com'
... ).add_path(
...     '_search'
... ).add_query_from(
...     [('q', 'repo:sigmavirus24/rfc3986'), ('sort', 'created_at:asc')]
... ).finalize().unsplit())
https://search.example.com/_search?q=repo%3Asigmavirus24%2Frfc3986&sort=created_at
↪%3Aasc
```

Finally, we provide a way to add a fragment to a URL. Let's build up a URL to view the section of the RFC that refers to fragments:

```
>>> from rfc3986 import builder
>>> print(builder.URIBuilder().add_scheme(
...     'https'
... ).add_host(
...     'tools.ietf.org'
... ).add_path(
```

(continues on next page)

(continued from previous page)

```
...     '/html/rfc3986'
... ).add_fragment(
...     'section-3.5'
... ).finalize().unsplit())
https://tools.ietf.org/html/rfc3986#section-3.5
```

CHAPTER 2

API Reference

This section contains API documentation generated from the source code of `rfc3986`. If you're looking for an introduction to the module and how it can be utilized, please see *User Documentation* instead.

2.1 API Submodule

`rfc3986.api.`**`urlparse`**(*uri*, *encoding='utf-8'*)

Parse a given URI and return a ParseResult.

This is a partial replacement of the standard library's urlparse function.

Parameters

- **`uri`** (*str*) – The URI to be parsed.
- **`encoding`** (*str*) – The encoding of the string provided.

Returns A parsed URI

Return type `ParseResult`

`rfc3986.api.`**`uri_reference`**(*uri*, *encoding='utf-8'*)

Parse a URI string into a URIReference.

This is a convenience function. You could achieve the same end by using `URIReference.from_string(uri)`.

Parameters

- **`uri`** (*str*) – The URI which needs to be parsed into a reference.
- **`encoding`** (*str*) – The encoding of the string provided

Returns A parsed URI

Return type `URIReference`

rfc3986.api.**normalize_uri**(*uri*, *encoding='utf-8'*)
Normalize the given URI.

This is a convenience function. You could use either `uri_reference(uri).normalize().unsplit()` or `URIReference.from_string(uri).normalize().unsplit()` instead.

Parameters

- **uri** (*str*) – The URI to be normalized.
- **encoding** (*str*) – The encoding of the string provided

Returns The normalized URI.

Return type str

2.2 URI Builder Module

class rfc3986.builder.**URIBuilder**(*scheme=None*, *userinfo=None*, *host=None*, *port=None*, *path=None*, *query=None*, *fragment=None*)
Object to aid in building up a URI Reference from parts.

Note: This object should be instantiated by the user, but it's recommended that it is not provided with arguments. Instead, use the available method to populate the fields.

classmethod URIBuilder.**from_uri**(*reference*)
Initialize the URI builder from another URI.

Takes the given URI reference and creates a new URI builder instance populated with the values from the reference. If given a string it will try to convert it to a reference before constructing the builder.

URIBuilder.**add_scheme**(*scheme*)
Add a scheme to our builder object.

After normalizing, this will generate a new URIBuilder instance with the specified scheme and all other attributes the same.

```
>>> URIBuilder().add_scheme('HTTPS')
URIBuilder(scheme='https', userinfo=None, host=None, port=None,
        path=None, query=None, fragment=None)
```

URIBuilder.**add_credentials**(*username*, *password*)
Add credentials as the userinfo portion of the URI.

```
>>> URIBuilder().add_credentials('root', 's3crete')
URIBuilder(scheme=None, userinfo='root:s3crete', host=None,
        port=None, path=None, query=None, fragment=None)

>>> URIBuilder().add_credentials('root', None)
URIBuilder(scheme=None, userinfo='root', host=None,
        port=None, path=None, query=None, fragment=None)
```

URIBuilder.**add_host**(*host*)
Add hostname to the URI.

```
>>> URIBuilder().add_host('google.com')
URIBuilder(scheme=None, userinfo=None, host='google.com',
        port=None, path=None, query=None, fragment=None)
```

URIBuilder.**add_port**(*port*)

Add port to the URI.

```
>>> URIBuilder().add_port(80)
URIBuilder(scheme=None, userinfo=None, host=None, port='80',
        path=None, query=None, fragment=None)

>>> URIBuilder().add_port(443)
URIBuilder(scheme=None, userinfo=None, host=None, port='443',
        path=None, query=None, fragment=None)
```

URIBuilder.**add_path**(*path*)

Add a path to the URI.

```
>>> URIBuilder().add_path('sigmavirus24/rfc3985')
URIBuilder(scheme=None, userinfo=None, host=None, port=None,
        path='/sigmavirus24/rfc3986', query=None, fragment=None)

>>> URIBuilder().add_path('/checkout.php')
URIBuilder(scheme=None, userinfo=None, host=None, port=None,
        path='/checkout.php', query=None, fragment=None)
```

URIBuilder.**add_query_from**(*query_items*)

Generate and add a query a dictionary or list of tuples.

```
>>> URIBuilder().add_query_from({'a': 'b c'})
URIBuilder(scheme=None, userinfo=None, host=None, port=None,
        path=None, query='a=b+c', fragment=None)

>>> URIBuilder().add_query_from([('a', 'b c')])
URIBuilder(scheme=None, userinfo=None, host=None, port=None,
        path=None, query='a=b+c', fragment=None)
```

URIBuilder.**add_query**(*query*)

Add a pre-formated query string to the URI.

```
>>> URIBuilder().add_query('a=b&c=d')
URIBuilder(scheme=None, userinfo=None, host=None, port=None,
        path=None, query='a=b&c=d', fragment=None)
```

URIBuilder.**add_fragment**(*fragment*)

Add a fragment to the URI.

```
>>> URIBuilder().add_fragment('section-2.6.1')
URIBuilder(scheme=None, userinfo=None, host=None, port=None,
        path=None, query=None, fragment='section-2.6.1')
```

URIBuilder.**finalize**()

Create a URIReference from our builder.

```
>>> URIBuilder().add_scheme('https').add_host('github.com'
...     ).add_path('sigmavirus24/rfc3986').finalize().unsplit()
'https://github.com/sigmavirus24/rfc3986'
```

(continues on next page)

(continued from previous page)

```
>>> URIBuilder().add_scheme('https').add_host('github.com'
...     ).add_path('sigmavirus24/rfc3986').add_credentials(
...     'sigmavirus24', 'not-re@l').finalize().unsplit()
'https://sigmavirus24:not-re%40l@github.com/sigmavirus24/rfc3986'
```

2.3 URI Submodule

class rfc3986.uri.**URIReference**

Immutable object representing a parsed URI Reference.

Note: This class is not intended to be directly instantiated by the user.

This object exposes attributes for the following components of a URI:

- scheme
- authority
- path
- query
- fragment

scheme

The scheme that was parsed for the URI Reference. For example, `http`, `https`, `smtp`, `imap`, etc.

authority

Component of the URI that contains the user information, host, and port sub-components. For example, `google.com`, `127.0.0.1:5000`, `username@[::1]`, `username:password@example.com:443`, etc.

path

The path that was parsed for the given URI Reference. For example, `/`, `/index.php`, etc.

query

The query component for a given URI Reference. For example, `a=b`, `a=b%20c`, `a=b+c`, `a=b,c=d,e=%20f`, etc.

fragment

The fragment component of a URI. For example, `section-3.1`.

This class also provides extra attributes for easier access to information like the subcomponents of the authority component.

userinfo

The user information parsed from the authority.

host

The hostname, IPv4, or IPv6 address parsed from the authority.

port

The port parsed from the authority.

classmethod URIReference.**from_string**(*uri_string*, *encoding='utf-8'*)

Parse a URI reference from the given unicode URI string.

Parameters

- **uri_string** (*str*) – Unicode URI to be parsed into a reference.
- **encoding** (*str*) – The encoding of the string provided

Returns *URIReference* or subclass thereof

URIReference.**unsplit**()
Create a URI string from the components.

Returns The URI Reference reconstituted as a string.

Return type str

URIReference.**resolve_with**(*base_uri*, *strict=False*)
Use an absolute URI Reference to resolve this relative reference.

Assuming this is a relative reference that you would like to resolve, use the provided base URI to resolve it.

See http://tools.ietf.org/html/rfc3986#section-5 for more information.

Parameters **base_uri** – Either a string or URIReference. It must be an absolute URI or it will raise an exception.

Returns A new URIReference which is the result of resolving this reference using `base_uri`.

Return type *URIReference*

Raises **rfc3986.exceptions.ResolutionError** – If the `base_uri` is not an absolute URI.

URIReference.**copy_with**(*scheme=<object object>*, *authority=<object object>*, *path=<object object>*, *query=<object object>*, *fragment=<object object>*)
Create a copy of this reference with the new components.

Parameters

- **scheme** (*str*) – (optional) The scheme to use for the new reference.
- **authority** (*str*) – (optional) The authority to use for the new reference.
- **path** (*str*) – (optional) The path to use for the new reference.
- **query** (*str*) – (optional) The query to use for the new reference.
- **fragment** (*str*) – (optional) The fragment to use for the new reference.

Returns New URIReference with provided components.

Return type *URIReference*

URIReference.**normalize**()
Normalize this reference as described in Section 6.2.2.

This is not an in-place normalization. Instead this creates a new URIReference.

Returns A new reference object with normalized components.

Return type *URIReference*

URIReference.**is_absolute**()
Determine if this URI Reference is an absolute URI.

See http://tools.ietf.org/html/rfc3986#section-4.3 for explanation.

Returns `True` if it is an absolute URI, `False` otherwise.

Return type bool

URIReference.**authority_info**()
Return a dictionary with the `userinfo`, `host`, and `port`.

If the authority is not valid, it will raise a `InvalidAuthority` Exception.

Returns `{'userinfo': 'username:password', 'host': 'www.example.com', 'port': '80'}`

Return type dict

Raises **rfc3986.exceptions.InvalidAuthority** – If the authority is not `None` and can not be parsed.

2.3.1 Deprecated Methods

URIReference.**is_valid**(***kwargs*)
Determine if the URI is valid.

Deprecated since version 1.1.0: Use the *Validator* object instead.

Parameters

- **require_scheme** (*bool*) – Set to `True` if you wish to require the presence of the scheme component.
- **require_authority** (*bool*) – Set to `True` if you wish to require the presence of the authority component.
- **require_path** (*bool*) – Set to `True` if you wish to require the presence of the path component.
- **require_query** (*bool*) – Set to `True` if you wish to require the presence of the query component.
- **require_fragment** (*bool*) – Set to `True` if you wish to require the presence of the fragment component.

Returns `True` if the URI is valid. `False` otherwise.

Return type bool

URIReference.**authority_is_valid**(*require=False*)
Determine if the authority component is valid.

Deprecated since version 1.1.0: Use the *Validator* object instead.

Parameters **require** (*bool*) – Set to `True` to require the presence of this component.

Returns `True` if the authority is valid. `False` otherwise.

Return type bool

URIReference.**scheme_is_valid**(*require=False*)
Determine if the scheme component is valid.

Deprecated since version 1.1.0: Use the *Validator* object instead.

Parameters **require** (*str*) – Set to `True` to require the presence of this component.

Returns `True` if the scheme is valid. `False` otherwise.

Return type bool

URIReference.**path_is_valid**(*require=False*)
Determine if the path component is valid.

Deprecated since version 1.1.0: Use the *Validator* object instead.

Parameters **require** (*str*) – Set to `True` to require the presence of this component.

Returns `True` if the path is valid. `False` otherwise.

Return type bool

URIReference.**query_is_valid**(*require=False*)
Determine if the query component is valid.

Deprecated since version 1.1.0: Use the *Validator* object instead.

Parameters **require** (*str*) – Set to `True` to require the presence of this component.

Returns `True` if the query is valid. `False` otherwise.

Return type bool

URIReference.**fragment_is_valid**(*require=False*)
Determine if the fragment component is valid.

Deprecated since version 1.1.0: Use the Validator object instead.

Parameters **require** (*str*) – Set to `True` to require the presence of this component.

Returns `True` if the fragment is valid. `False` otherwise.

Return type bool

2.4 Validators Submodule

class rfc3986.validators.**Validator**
Object used to configure validation of all objects in rfc3986.

New in version 1.0.

Example usage:

```
>>> from rfc3986 import api, validators
>>> uri = api.uri_reference('https://github.com/')
>>> validator = validators.Validator().require_presence_of(
...     'scheme', 'host', 'path',
... ).allow_schemes(
...     'http', 'https',
... ).allow_hosts(
...     '127.0.0.1', 'github.com',
... )
>>> validator.validate(uri)
>>> invalid_uri = rfc3986.uri_reference('imap://mail.google.com')
>>> validator.validate(invalid_uri)
Traceback (most recent call last):
...
rfc3986.exceptions.MissingComponentError: ('path was required but
missing', URIReference(scheme=u'imap', authority=u'mail.google.com',
path=None, query=None, fragment=None), ['path'])
```

Validator.**allow_schemes**(**schemes*)
Require the scheme to be one of the provided schemes.

New in version 1.0.

Parameters **schemes** – Schemes, without `://` that are allowed.

Returns The validator instance.

Return type *Validator*

Validator.**allow_hosts**(**hosts*)
Require the host to be one of the provided hosts.

New in version 1.0.

Parameters **hosts** – Hosts that are allowed.

Returns The validator instance.

Return type *Validator*

Validator.**allow_ports**(**ports*)
Require the port to be one of the provided ports.

New in version 1.0.

Parameters **ports** – Ports that are allowed.

Returns The validator instance.

Return type *Validator*

Validator.**allow_use_of_password**()
Allow passwords to be present in the URI.

New in version 1.0.

Returns The validator instance.

Return type *Validator*

Validator.**check_validity_of**(**components*)
Check the validity of the components provided.

This can be specified repeatedly.

New in version 1.1.

Parameters **components** – Names of components from `Validator.COMPONENT_NAMES`.

Returns The validator instance.

Return type *Validator*

Validator.**forbid_use_of_password**()
Prevent passwords from being included in the URI.

New in version 1.0.

Returns The validator instance.

Return type *Validator*

Validator.**require_presence_of**(**components*)
Require the components provided.

This can be specified repeatedly.

New in version 1.0.

Parameters **components** – Names of components from `Validator.COMPONENT_NAMES`.

Returns The validator instance.

Return type *Validator*

`Validator.`**validate**(*uri*)
Check a URI for conditions specified on this validator.

New in version 1.0.

Parameters **uri** (`rfc3986.uri.URIReference`) – Parsed URI to validate.

Raises

- **MissingComponentError** – When a required component is missing.
- **UnpermittedComponentError** – When a component is not one of those allowed.
- **PasswordForbidden** – When a password is present in the userinfo component but is not permitted by configuration.
- **InvalidComponentsError** – When a component was found to be invalid.

2.5 IRI Submodule

class `rfc3986.iri.`**IRIReference**
Immutable object representing a parsed IRI Reference.

Can be encoded into an URIReference object via the procedure specified in RFC 3987 Section 3.1

Note: The IRI submodule is a new interface and may possibly change in the future. Check for changes to the interface when upgrading.

`IRIReference.`**encode**(*idna_encoder=None*)
Encode an IRIReference into a URIReference instance.

If the `idna` module is installed or the `rfc3986[idna]` extra is used then unicode characters in the IRI host component will be encoded with IDNA2008.

Parameters **idna_encoder** – Function that encodes each part of the host component If not given will raise an exception if the IRI contains a host component.

Return type *uri.URIReference*

Returns A URI reference

classmethod `IRIReference.`**from_string**(*iri_string*, *encoding='utf-8'*)
Parse a IRI reference from the given unicode IRI string.

Parameters

- **iri_string** (*str*) – Unicode IRI to be parsed into a reference.
- **encoding** (*str*) – The encoding of the string provided

Returns *IRIReference* or subclass thereof

`IRIReference.`**unsplit**()
Create a URI string from the components.

Returns The URI Reference reconstituted as a string.

Return type str

IRIReference.**resolve_with**(*base_uri*, *strict=False*)

Use an absolute URI Reference to resolve this relative reference.

Assuming this is a relative reference that you would like to resolve, use the provided base URI to resolve it.

See http://tools.ietf.org/html/rfc3986#section-5 for more information.

Parameters **base_uri** – Either a string or URIReference. It must be an absolute URI or it will raise an exception.

Returns A new URIReference which is the result of resolving this reference using `base_uri`.

Return type `URIReference`

Raises **rfc3986.exceptions.ResolutionError** – If the `base_uri` is not an absolute URI.

IRIReference.**copy_with**(*scheme=<object object>*, *authority=<object object>*, *path=<object object>*, *query=<object object>*, *fragment=<object object>*)

Create a copy of this reference with the new components.

Parameters

- **scheme** (*str*) – (optional) The scheme to use for the new reference.
- **authority** (*str*) – (optional) The authority to use for the new reference.
- **path** (*str*) – (optional) The path to use for the new reference.
- **query** (*str*) – (optional) The query to use for the new reference.
- **fragment** (*str*) – (optional) The fragment to use for the new reference.

Returns New URIReference with provided components.

Return type *URIReference*

IRIReference.**is_absolute**()

Determine if this URI Reference is an absolute URI.

See http://tools.ietf.org/html/rfc3986#section-4.3 for explanation.

Returns `True` if it is an absolute URI, `False` otherwise.

Return type bool

IRIReference.**authority_info**()

Return a dictionary with the `userinfo`, `host`, and `port`.

If the authority is not valid, it will raise a `InvalidAuthority` Exception.

Returns `{'userinfo': 'username:password', 'host': 'www.example.com', 'port': '80'}`

Return type dict

Raises **rfc3986.exceptions.InvalidAuthority** – If the authority is not `None` and can not be parsed.

2.6 Miscellaneous Submodules

There are several submodules in `rfc3986` that are not meant to be exposed to users directly but which are valuable to document, regardless.

`rfc3986.misc.`**`UseExisting`**
A sentinel object to make certain APIs simpler for users.

The *`rfc3986.abnf_regexp`* module contains the regular expressions written from the RFC's ABNF. The *`rfc3986.misc`* module contains compiled regular expressions from *`rfc3986.abnf_regexp`* and previously contained those regular expressions.

`rfc3986.abnf_regexp.`**`GEN_DELIMS`**

`rfc3986.abnf_regexp.`**`GENERIC_DELIMITERS`**
The string containing all of the generic delimiters as defined on page 13.

`rfc3986.abnf_regexp.`**`GENERIC_DELIMITERS_SET`**
`rfc3986.abnf_regexp.GEN_DELIMS` represented as a `set`.

`rfc3986.abnf_regexp.`**`SUB_DELIMS`**

`rfc3986.abnf_regexp.`**`SUB_DELIMITERS`**
The string containing all of the 'sub' delimiters as defined on page 13.

`rfc3986.abnf_regexp.`**`SUB_DELIMITERS_SET`**
`rfc3986.abnf_regexp.SUB_DELIMS` represented as a `set`.

`rfc3986.abnf_regexp.`**`SUB_DELIMITERS_RE`**
`rfc3986.abnf_regexp.SUB_DELIMS` with the `*` escaped for use in regular expressions.

`rfc3986.abnf_regexp.`**`RESERVED_CHARS_SET`**
A `set` constructed of *`GEN_DELIMS`* and *`SUB_DELIMS`*. This union is defined on page 13.

`rfc3986.abnf_regexp.`**`ALPHA`**
The string of upper- and lower-case letters in USASCII.

`rfc3986.abnf_regexp.`**`DIGIT`**
The string of digits 0 through 9.

`rfc3986.abnf_regexp.`**`UNRESERVED`**

`rfc3986.abnf_regexp.`**`UNRESERVED_CHARS`**
The string of unreserved characters defined in **RFC 3986#section-2.3**.

`rfc3986.abnf_regexp.`**`UNRESERVED_CHARS_SET`**
`rfc3986.abnf_regexp.UNRESERVED_CHARS` represented as a `set`.

`rfc3986.abnf_regexp.`**`NON_PCT_ENCODED_SET`**
The non-percent encoded characters represented as a `set`.

`rfc3986.abnf_regexp.`**`UNRESERVED_RE`**
Optimized regular expression for unreserved characters.

`rfc3986.abnf_regexp.`**`SCHEME_RE`**
Stricter regular expression to match and validate the scheme part of a URI.

`rfc3986.abnf_regexp.`**`COMPONENT_PATTERN_DICT`**
Dictionary with regular expressions to match various components in a URI. Except for *`rfc3986.abnf_regexp.SCHEME_RE`*, all patterns are from **RFC 3986#appendix-B**.

rfc3986.abnf_regexp.**URL_PARSING_RE**
Regular expression compposed from the components in *rfc3986.abnf_regexp.COMPONENT_PATTERN_DICT*.

rfc3986.abnf_regexp.**HEXDIG_RE**
Hexadecimal characters used in each piece of an IPv6 address. See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**LS32_RE**
Lease significant 32 bits of an IPv6 address. See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**REG_NAME**

rfc3986.abnf_regexp.**REGULAR_NAME_RE**
The pattern for a regular name, e.g., `www.google.com`, `api.github.com`. See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**IPv4_RE**
The pattern for an IPv4 address, e.g., `192.168.255.255`. See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**IPv6_RE**
The pattern for an IPv6 address, e.g., `::1`. See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**IPv_FUTURE_RE**
A regular expression to parse out IPv Futures. See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**IP_LITERAL_RE**
Pattern to match IPv6 addresses and IPv Future addresses. See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**HOST_RE**

rfc3986.abnf_regexp.**HOST_PATTERN**
Pattern to match and validate the host piece of an authority. This is composed of

- *rfc3986.abnf_regexp.REG_NAME*
- *rfc3986.abnf_regexp.IPv4_RE*
- *rfc3986.abnf_regexp.IP_LITERAL_RE*

See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**USERINFO_RE**
Pattern to match and validate the user information portion of an authority component.

See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**PORT_RE**
Pattern to match and validate the port portion of an authority component.

See **RFC 3986#section-3.2.2**.

rfc3986.abnf_regexp.**PCT_ENCODED**

rfc3986.abnf_regexp.**PERCENT_ENCODED**
Regular expression to match percent encoded character values.

rfc3986.abnf_regexp.**PCHAR**
Regular expression to match printable characters.

rfc3986.abnf_regexp.**PATH_RE**
Regular expression to match and validate the path component of a URI.

See **RFC 3986#section-3.3**.

rfc3986.abnf_regexp.**PATH_EMPTY**

rfc3986.abnf_regexp.**PATH_ROOTLESS**

rfc3986.abnf_regexp.**PATH_NOSCHEME**

rfc3986.abnf_regexp.**PATH_ABSOLUTE**

rfc3986.abnf_regexp.**PATH_ABEMPTY**
Components of the *rfc3986.abnf_regexp.PATH_RE*.

See **RFC 3986#section-3.3**.

rfc3986.abnf_regexp.**QUERY_RE**
Regular expression to parse and validate the query component of a URI.

rfc3986.abnf_regexp.**FRAGMENT_RE**
Regular expression to parse and validate the fragment component of a URI.

rfc3986.abnf_regexp.**RELATIVE_PART_RE**
Regular expression to parse the relative URI when resolving URIs.

rfc3986.abnf_regexp.**HIER_PART_RE**
The hierarchical part of a URI. This regular expression is used when resolving relative URIs.

See **RFC 3986#section-3**.

rfc3986.misc.**URI_MATCHER**
Compiled version of rfc3986.abnf_regexp.URL_PARSING_RE.

rfc3986.misc.**SUBAUTHORITY_MATCHER**
Compiled compilation of rfc3986.abnf_regexp.USERINFO_RE, rfc3986.abnf_regexp.HOST_PATTERN, rfc3986.abnf_regexp.PORT_RE.

rfc3986.misc.**SCHEME_MATCHER**
Compiled version of rfc3986.abnf_regexp.SCHEME_RE.

rfc3986.misc.**IPv4_MATCHER**
Compiled version of rfc3986.abnf_regexp.IPv4_RE.

rfc3986.misc.**PATH_MATCHER**
Compiled version of rfc3986.abnf_regexp.PATH_RE.

rfc3986.misc.**QUERY_MATCHER**
Compiled version of rfc3986.abnf_regexp.QUERY_RE.

rfc3986.misc.**RELATIVE_REF_MATCHER**
Compiled compilation of rfc3986.abnf_regexp.SCHEME_RE, rfc3986.abnf_regexp.HIER_PART_RE, rfc3986.abnf_regexp.QUERY_RE.

CHAPTER 3

Release Notes and History

All of the release notes that have been recorded for `rfc3986` are organized here with the newest releases first.

3.1 1.x Release Series

3.1.1 1.3.0 – 2020-04-07

Security

- Prevent users from receiving an invalid authority parsed from a malicious URL. Previously we did not stop parsing the authority section at the first backslash (`\\`) character. As a result, it was possible to trick our parser into parsing up to the first forward-slash (`/`) and thus generating an invalid authority.

  See also GitHub pr-64 and the blog post that sparked this change

Bug Fixes and Features

- Add `from_uri` to `URIBuilder` to allow creation of a `URIBuilder` from an existing URI.

  See also GitHub pr-63

- Fix a typographical error in our documentation.

  See also GitHub pr-61

3.1.2 1.3.2 – 2019-05-13

- Remove unnecessary IRI-flavored matchers from `rfc3986.misc` to speed up import time on resource-constrained systems.

  See also GitHub #55

3.1.3 1.3.1 – 2019-04-23

- Only apply IDNA-encoding when there are characters outside of the ASCII character set.

  See also GitHub #52

3.1.4 1.3.0 – 2019-04-20

- Add the `IRIReference` class which parses data according to RFC 3987 and encodes into an `URIReference`.

  See also GitHub #50

3.1.5 1.2.0 – 2018-12-04

- Attempt to detect percent-encoded URI components and encode `%` characters if required.

  See also GitHub #38

- Allow percent-encoded bytes within host.

  See also GitHub #39

- Correct the IPv6 regular expression by adding a missing variation.
- Fix hashing for URIReferences on Python 3.

  See also GitHub !35

3.1.6 1.1.0 – 2017-07-18

- Correct the regular expression for the User Information sub-component of the Authority Component.

  See also GitHub #26

- *`check_validity_of()`* to the *`Validator`* class. See *Validating URIs* documentation for more information.

3.1.7 1.0.0 – 2017-05-10

- Add support for **RFC 6874** - Zone Identifiers in IPv6 Addresses

  See also issue #2

- Add a more flexible and usable validation framework. See our documentation for more information.
- Add an object to aid in building new URIs from scratch. See our documentation for more information.
- Add real documentation for the entire module.
- Add separate submodule with documented regular expression strings for the collected ABNF.
- Allow `None` to be used to eliminate components via `copy_with` for URIs and ParseResults.
- Move release history into our documentation.

3.2 0.x Release Series

3.2.1 0.4.2 – 2016-08-22

- Avoid parsing an string with just an IPv6 address as having a scheme of `[`.

3.2.2 0.4.1 – 2016-08-22

- Normalize URIs constructed using `ParseResult.from_parts` and `ParseResultBytes.from_parts`

3.2.3 0.4.0 – 2016-08-20

- Add `ParseResult.from_parts` and `ParseResultBytes.from_parts` class methods to easily create a ParseResult
- When using regular expressions, use `[0-9]` instead of `\d` to avoid finding ports with "numerals" that are not valid in a port

3.2.4 0.3.1 – 2015-12-15

- Preserve empty query strings during normalization

3.2.5 0.3.0 – 2015-10-20

- Read README and HISTORY files using the appropriate codec so rfc3986 can be installed on systems with locale's other than utf-8 (specifically C)
- Replace the standard library's urlparse behaviour

3.2.6 0.2.2 – 2015-05-27

- Update the regular name regular expression to accept all of the characters allowed in the RFC. Closes bug #11 (Thanks Viktor Haag). Previously URIs similar to "http://http-bin.org" would be considered invalid.

3.2.7 0.2.1 – 2015-03-20

- Check that the bytes of an IPv4 Host Address are within the valid range. Otherwise, URIs like "http://256.255.255.0/v1/resource" are considered valid.
- Add 6 to the list of unreserved characters. It was previously missing. Closes bug #9

3.2.8 0.2.0 – 2014-06-30

- Add support for requiring components during validation. This includes adding parameters `require_scheme`, `require_authority`, `require_path`, `require_path`, `require_query`, and `require_fragment` to `rfc3986.is_valid_uri` and `URIReference#is_valid`.

3.2.9 0.1.0 – 2014-06-27

- Initial Release includes validation and normalization of URIs

Python Module Index

r

Index

R

S

U

V