w3lib Documentation

Release 1.22.0

w3lib developers

Jun 16, 2022

Contents

CHAPTER 1

Overview

This is a Python library of web-related functions, such as:

- remove comments, or tags from HTML snippets
- extract base url from HTML snippets
- translate entities on HTML strings
- convert raw HTTP headers to dicts and vice-versa
- construct HTTP auth header
- converting HTML pages to unicode
- sanitize urls (like browsers do)
- extract arguments from urls

CHAPTER 2

Modules

2.1 w3lib Package

2.1.1 `encoding` Module

Functions for handling encoding of web pages

`w3lib.encoding.`**`html_body_declared_encoding`**(*html_body_str: Union[str, bytes]*) → Optional[str]

Return the encoding specified in meta tags in the html body, or `None` if no suitable encoding was found

```
>>> import w3lib.encoding
>>> w3lib.encoding.html_body_declared_encoding(
... """<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
...      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
... <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
... <head>
...     <title>Some title</title>
...     <meta http-equiv="content-type" content="text/html;charset=utf-8" />
... </head>
... <body>
... ...
... </body>
... </html>""")
'utf-8'
>>>
```

`w3lib.encoding.`**`html_to_unicode`**(*content_type_header: Optional[str], html_body_str: bytes, default_encoding: str = 'utf8', auto_detect_fun: Optional[Callable[[bytes], str]] = None*) → Tuple[str, str]

Convert raw html bytes to unicode

This attempts to make a reasonable guess at the content encoding of the html body, following a similar process to a web browser.

It will try in order:

- http content type header
- BOM (byte-order mark)
- meta or xml tag declarations
- auto-detection, if the *auto_detect_fun* keyword argument is not `None`
- default encoding in keyword arg (which defaults to utf8)

If an encoding other than the auto-detected or default encoding is used, overrides will be applied, converting some character encodings to more suitable alternatives.

If a BOM is found matching the encoding, it will be stripped.

The *auto_detect_fun* argument can be used to pass a function that will sniff the encoding of the text. This function must take the raw text as an argument and return the name of an encoding that python can process, or None. To use chardet, for example, you can define the function as:

```
auto_detect_fun=lambda x: chardet.detect(x).get('encoding')
```

or to use UnicodeDammit (shipped with the BeautifulSoup library):

```
auto_detect_fun=lambda x: UnicodeDammit(x).originalEncoding
```

If the locale of the website or user language preference is known, then a better default encoding can be supplied.

If *content_type_header* is not present, `None` can be passed signifying that the header was not present.

This method will not fail, if characters cannot be converted to unicode, `\\ufffd` (the unicode replacement character) will be inserted instead.

Returns a tuple of `(<encoding used>, <unicode_string>)`

Examples:

```
>>> import w3lib.encoding
>>> w3lib.encoding.html_to_unicode(None,
... b"""<!DOCTYPE html>
... <head>
... <meta charset="UTF-8" />
... <meta name="viewport" content="width=device-width" />
... <title>Creative Commons France</title>
... <link rel='canonical' href='http://creativecommons.fr/' />
... <body>
... <p>Creative Commons est une organisation \xc3\xa0 but non lucratif
... qui a pour dessein de faciliter la diffusion et le partage des oeuvres
... tout en accompagnant les nouvelles pratiques de cr\xc3\xa9ation \xc3\xa0␣
↪1\xe2\x80\x99\xc3\xa8re numerique.</p>
... </body>
... </html>""")
('utf-8', '<!DOCTYPE html>\n<head>\n<meta charset="UTF-8" />\n<meta name="viewport
↪" content="width=device-width" />\n<title>Creative Commons France</title>\n
↪<link rel=\'canonical\' href=\'http://creativecommons.fr/\' />\n<body>\n<p>
↪Creative Commons est une organisation \xe0 but non lucratif\nqui a pour dessein␣
↪de faciliter la diffusion et le partage des oeuvres\ntout en accompagnant les␣
↪nouvelles pratiques de cr\xe9ation \xe0 l\u2019\xe8re numerique.</p>\n</body>\n
↪</html>')
>>>
```

w3lib.encoding.**http_content_type_encoding**(*content_type: Optional[str]*) → Optional[str]
Extract the encoding in the content-type header

```
>>> import w3lib.encoding
>>> w3lib.encoding.http_content_type_encoding("Content-Type: text/html;␣
↪charset=ISO-8859-4")
'iso8859-4'
```

w3lib.encoding.**read_bom**(*data: bytes*) → Union[Tuple[None, None], Tuple[str, bytes]]
Read the byte order mark in the text, if present, and return the encoding represented by the BOM and the BOM.

If no BOM can be detected, `(None, None)` is returned.

```
>>> import w3lib.encoding
>>> w3lib.encoding.read_bom(b'\xfe\xff\x6c\x34')
('utf-16-be', '\xfe\xff')
>>> w3lib.encoding.read_bom(b'\xff\xfe\x34\x6c')
('utf-16-le', '\xff\xfe')
>>> w3lib.encoding.read_bom(b'\x00\x00\xfe\xff\x00\x00\x6c\x34')
('utf-32-be', '\x00\x00\xfe\xff')
>>> w3lib.encoding.read_bom(b'\xff\xfe\x00\x00\x34\x6c\x00\x00')
('utf-32-le', '\xff\xfe\x00\x00')
>>> w3lib.encoding.read_bom(b'\x01\x02\x03\x04')
(None, None)
>>>
```

w3lib.encoding.**resolve_encoding**(*encoding_alias: str*) → Optional[str]
Return the encoding that *encoding_alias* maps to, or `None` if the encoding cannot be interpreted

```
>>> import w3lib.encoding
>>> w3lib.encoding.resolve_encoding('latin1')
'cp1252'
>>> w3lib.encoding.resolve_encoding('gb_2312-80')
'gb18030'
>>>
```

w3lib.encoding.**to_unicode**(*data_str: bytes*, *encoding: str*) → str
Convert a str object to unicode using the encoding given

Characters that cannot be converted will be converted to `\ufffd` (the unicode replacement character).

2.1.2 `html` Module

Functions for dealing with markup text

w3lib.html.**get_base_url**(*text: AnyStr*, *baseurl: Union[str, bytes] = ''*, *encoding: str = 'utf-8'*) → str
Return the base url if declared in the given HTML *text*, relative to the given base url.

If no base url is found, the given *baseurl* is returned.

w3lib.html.**get_meta_refresh**(*text: AnyStr*, *baseurl: str = ''*, *encoding: str = 'utf-8'*, *ignore_tags: Iterable[str] = ('script', 'noscript')*) → Tuple[Optional[float], Optional[str]]
Return the http-equiv parameter of the HTML meta element from the given HTML text and return a tuple `(interval, url)` where interval is an integer containing the delay in seconds (or zero if not present) and url is a string with the absolute url to redirect.

If no meta redirect is found, `(None, None)` is returned.

w3lib.html.**remove_comments**(*text: AnyStr*, *encoding: Optional[str] = None*) → str

Remove HTML Comments.

```
>>> import w3lib.html
>>> w3lib.html.remove_comments(b"test <!--textcoment--> whatever")
'test  whatever'
>>>
```

w3lib.html.**remove_tags**(*text: AnyStr*, *which_ones: Iterable[str] = ()*, *keep: Iterable[str] = ()*, *encoding: Optional[str] = None*) → str

Remove HTML Tags only.

which_ones and *keep* are both tuples, there are four cases:

which_ones	keep	what it does
not empty	empty	remove all tags in which_ones
empty	**not empty**	remove all tags except the ones in keep
empty	empty	remove all tags
not empty	**not empty**	not allowed

Remove all tags:

```
>>> import w3lib.html
>>> doc = '<div><p><b>This is a link:</b> <a href="http://www.example.com">example
↪</a></p></div>'
>>> w3lib.html.remove_tags(doc)
'This is a link: example'
>>>
```

Keep only some tags:

```
>>> w3lib.html.remove_tags(doc, keep=('div',))
'<div>This is a link: example</div>'
>>>
```

Remove only specific tags:

```
>>> w3lib.html.remove_tags(doc, which_ones=('a','b'))
'<div><p>This is a link: example</p></div>'
>>>
```

You can't remove some and keep some:

```
>>> w3lib.html.remove_tags(doc, which_ones=('a',), keep=('p',))
Traceback (most recent call last):
    ...
ValueError: Cannot use both which_ones and keep
>>>
```

w3lib.html.**remove_tags_with_content**(*text: AnyStr*, *which_ones: Iterable[str] = ()*, *encoding: Optional[str] = None*) → str

Remove tags and their content.

which_ones is a tuple of which tags to remove including their content. If is empty, returns the string unmodified.

```
>>> import w3lib.html
>>> doc = '<div><p><b>This is a link:</b> <a href="http://www.example.com">example
↪</a></p></div>'
```

(continues on next page)

(continued from previous page)

```
>>> w3lib.html.remove_tags_with_content(doc, which_ones=('b',))
'<div><p> <a href="http://www.example.com">example</a></p></div>'
>>>
```

w3lib.html.**replace_entities**(*text: AnyStr*, *keep: Iterable[str] = ()*, *remove_illegal: bool = True*, *encoding: str = 'utf-8'*) → str

Remove entities from the given *text* by converting them to their corresponding unicode character.

text can be a unicode string or a byte string encoded in the given *encoding* (which defaults to 'utf-8').

If *keep* is passed (with a list of entity names) those entities will be kept (they won't be removed).

It supports both numeric entities (`&#nnnn;` and `&#hhhh;`) and named entities (such as ` ` or `>`).

If *remove_illegal* is `True`, entities that can't be converted are removed. If *remove_illegal* is `False`, entities that can't be converted are kept "as is". For more information see the tests.

Always returns a unicode string (with the entities removed).

```
>>> import w3lib.html
>>> w3lib.html.replace_entities(b'Price: &pound;100')
'Price: \xa3100'
>>> print(w3lib.html.replace_entities(b'Price: &pound;100'))
Price: £100
>>>
```

w3lib.html.**replace_escape_chars**(*text: AnyStr*, *which_ones: Iterable[str] = ('\n', '\t', '\r')*, *replace_by: Union[str, bytes] = ''*, *encoding: Optional[str] = None*) → str

Remove escape characters.

which_ones is a tuple of which escape characters we want to remove. By default removes `\n`, `\t`, `\r`.

replace_by is the string to replace the escape characters by. It defaults to `''`, meaning the escape characters are removed.

w3lib.html.**replace_tags**(*text: AnyStr*, *token: str = ''*, *encoding: Optional[str] = None*) → str

Replace all markup tags found in the given *text* by the given token. By default *token* is an empty string so it just removes all tags.

text can be a unicode string or a regular string encoded as *encoding* (or `'utf-8'` if *encoding* is not given.)

Always returns a unicode string.

Examples:

```
>>> import w3lib.html
>>> w3lib.html.replace_tags('This text contains <a>some tag</a>')
'This text contains some tag'
>>> w3lib.html.replace_tags('<p>Je ne parle pas <b>fran\xe7ais</b></p>', ' -- ',
↪'latin-1')
' -- Je ne parle pas  -- fran\xe7ais --  -- '
>>>
```

w3lib.html.**strip_html5_whitespace**(*text: str*) → str

Strip all leading and trailing space characters (as defined in https://www.w3.org/TR/html5/infrastructure.html#space-character).

Such stripping is useful e.g. for processing HTML element attributes which contain URLs, like `href`, `src` or form `action` - HTML5 standard defines them as "valid URL potentially surrounded by spaces" or "valid non-empty URL potentially surrounded by spaces".

```
>>> strip_html5_whitespace(' hello\n')
'hello'
```

w3lib.html.**unquote_markup**(*text: AnyStr*, *keep: Iterable[str] = ()*, *remove_illegal: bool = True*, *encoding: Optional[str] = None*) → str

This function receives markup as a text (always a unicode string or a UTF-8 encoded string) and does the following:

1. **removes entities (except the ones in *keep*) from any part of it** that is not inside a CDATA
2. searches for CDATAs and extracts their text (if any) without modifying it.
3. removes the found CDATAs

2.1.3 http Module

w3lib.http.**basic_auth_header**(*username: AnyStr*, *password: AnyStr*, *encoding: str = 'ISO-8859-1'*) → bytes

Return an *Authorization* header field value for HTTP Basic Access Authentication (RFC 2617)

```
>>> import w3lib.http
>>> w3lib.http.basic_auth_header('someuser', 'somepass')
'Basic c29tZXVzZXI6c29tZXBhc3M='
```

w3lib.http.**headers_dict_to_raw**(*headers_dict: Optional[Mapping[bytes, Union[Any, Sequence[T_co]]]]*) → Optional[bytes]

Returns a raw HTTP headers representation of headers

For example:

```
>>> import w3lib.http
>>> w3lib.http.headers_dict_to_raw({b'Content-type': b'text/html', b'Accept': b
↪'gzip'}) # doctest: +SKIP
'Content-type: text/html\\r\\nAccept: gzip'
>>>
```

Note that keys and values must be bytes.

Argument is `None` (returns `None`):

```
>>> w3lib.http.headers_dict_to_raw(None)
>>>
```

w3lib.http.**headers_raw_to_dict**(*headers_raw: Optional[bytes]*) → Optional[MutableMapping[bytes, List[bytes]]]

Convert raw headers (single multi-line bytestring) to a dictionary.

For example:

```
>>> import w3lib.http
>>> w3lib.http.headers_raw_to_dict(b"Content-type: text/html\n\rAccept: gzip\n\n
↪")    # doctest: +SKIP
{'Content-type': ['text/html'], 'Accept': ['gzip']}
```

Incorrect input:

```
>>> w3lib.http.headers_raw_to_dict(b"Content-typt gzip\n\n")
{}
>>>
```

Argument is `None` (return `None`):

```
>>> w3lib.http.headers_raw_to_dict(None)
>>>
```

2.1.4 `url` Module

This module contains general purpose URL functions not found in the standard library.

class w3lib.url.**ParseDataURIResult**
Named tuple returned by *parse_data_uri()*.

data
Data, decoded if it was encoded in base64 format.

media_type
MIME type type and subtype, separated by / (e.g. `"text/plain"`).

media_type_parameters
MIME type parameters (e.g. `{"charset": "US-ASCII"}`).

w3lib.url.**add_or_replace_parameter**(*url: str*, *name: str*, *new_value: str*) → str
Add or remove a parameter to a given url

```
>>> import w3lib.url
>>> w3lib.url.add_or_replace_parameter('http://www.example.com/index.php', 'arg',
↪'v')
'http://www.example.com/index.php?arg=v'
>>> w3lib.url.add_or_replace_parameter('http://www.example.com/index.php?arg1=v1&
↪arg2=v2&arg3=v3', 'arg4', 'v4')
'http://www.example.com/index.php?arg1=v1&arg2=v2&arg3=v3&arg4=v4'
>>> w3lib.url.add_or_replace_parameter('http://www.example.com/index.php?arg1=v1&
↪arg2=v2&arg3=v3', 'arg3', 'v3new')
'http://www.example.com/index.php?arg1=v1&arg2=v2&arg3=v3new'
>>>
```

w3lib.url.**add_or_replace_parameters**(*url: str*, *new_parameters: Dict[str, str]*) → str
Add or remove a parameters to a given url

```
>>> import w3lib.url
>>> w3lib.url.add_or_replace_parameters('http://www.example.com/index.php', {'arg
↪': 'v'})
'http://www.example.com/index.php?arg=v'
>>> args = {'arg4': 'v4', 'arg3': 'v3new'}
>>> w3lib.url.add_or_replace_parameters('http://www.example.com/index.php?arg1=v1&
↪arg2=v2&arg3=v3', args)
'http://www.example.com/index.php?arg1=v1&arg2=v2&arg3=v3new&arg4=v4'
>>>
```

w3lib.url.**any_to_uri**(*uri_or_path: str*) → str
If given a path name, return its File URI, otherwise return it unmodified

w3lib.url.**canonicalize_url**(*url: Union[str, bytes, urllib.parse.ParseResult]*, *keep_blank_values: bool = True*, *keep_fragments: bool = False*, *encoding: Optional[str] = None*) → str
Canonicalize the given url by applying the following procedures:

- sort query arguments, first by key, then by value

- percent encode paths ; non-ASCII characters are percent-encoded using UTF-8 (RFC-3986)
- percent encode query arguments ; non-ASCII characters are percent-encoded using passed *encoding* (UTF-8 by default)
- normalize all spaces (in query arguments) '+' (plus symbol)
- normalize percent encodings case (%2f -> %2F)
- remove query arguments with blank values (unless *keep_blank_values* is True)
- remove fragments (unless *keep_fragments* is True)

The url passed can be bytes or unicode, while the url returned is always a native str (bytes in Python 2, unicode in Python 3).

```
>>> import w3lib.url
>>>
>>> # sorting query arguments
>>> w3lib.url.canonicalize_url('http://www.example.com/do?c=3&b=5&b=2&a=50')
'http://www.example.com/do?a=50&b=2&b=5&c=3'
>>>
>>> # UTF-8 conversion + percent-encoding of non-ASCII characters
>>> w3lib.url.canonicalize_url('http://www.example.com/r\u00e9sum\u00e9')
'http://www.example.com/r%C3%A9sum%C3%A9'
>>>
```

For more examples, see the tests in *tests/test_url.py*.

`w3lib.url.`**`file_uri_to_path`**(*uri: str*) → str
Convert File URI to local filesystem path according to: http://en.wikipedia.org/wiki/File_URI_scheme

`w3lib.url.`**`parse_data_uri`**(*uri: Union[str, bytes]*) → w3lib.url.ParseDataURIResult
Parse a data: URI into *ParseDataURIResult*.

`w3lib.url.`**`path_to_file_uri`**(*path: str*) → str
Convert local filesystem path to legal File URIs as described in: http://en.wikipedia.org/wiki/File_URI_scheme

`w3lib.url.`**`safe_download_url`**(*url: Union[str, bytes], encoding: str = 'utf8', path_encoding: str = 'utf8'*) → str
Make a url for download. This will call safe_url_string and then strip the fragment, if one exists. The path will be normalised.

If the path is outside the document root, it will be changed to be within the document root.

`w3lib.url.`**`safe_url_string`**(*url: Union[str, bytes], encoding: str = 'utf8', path_encoding: str = 'utf8', quote_path: bool = True*) → str
Convert the given URL into a legal URL by escaping unsafe characters according to RFC-3986. Also, ASCII tabs and newlines are removed as per https://url.spec.whatwg.org/#url-parsing.

If a bytes URL is given, it is first converted to *str* using the given encoding (which defaults to 'utf-8'). If quote_path is True (default), path_encoding ('utf-8' by default) is used to encode URL path component which is then quoted. Otherwise, if quote_path is False, path component is not encoded or quoted. Given encoding is used for query string or form data.

When passing an encoding, you should use the encoding of the original page (the page from which the URL was extracted from).

Calling this function on an already "safe" URL will return the URL unmodified.

Always returns a native *str* (bytes in Python2, unicode in Python3).

w3lib.url.**url_query_cleaner**(*url: Union[str, bytes], parameterlist: Union[str, bytes, Sequence[Union[str, bytes]]] = (), sep: str = '&', kvsep: str = '=', remove: bool = False, unique: bool = True, keep_fragments: bool = False*) → str

Clean URL arguments leaving only those passed in the parameterlist keeping order

```
>>> import w3lib.url
>>> w3lib.url.url_query_cleaner("product.html?id=200&foo=bar&name=wired", ('id',))
'product.html?id=200'
>>> w3lib.url.url_query_cleaner("product.html?id=200&foo=bar&name=wired", ['id',
↪'name'])
'product.html?id=200&name=wired'
>>>
```

If *unique* is `False`, do not remove duplicated keys

```
>>> w3lib.url.url_query_cleaner("product.html?d=1&e=b&d=2&d=3&other=other", ['d'],
↪ unique=False)
'product.html?d=1&d=2&d=3'
>>>
```

If *remove* is `True`, leave only those **not in parameterlist**.

```
>>> w3lib.url.url_query_cleaner("product.html?id=200&foo=bar&name=wired", ['id'],␣
↪remove=True)
'product.html?foo=bar&name=wired'
>>> w3lib.url.url_query_cleaner("product.html?id=2&foo=bar&name=wired", ['id',
↪'foo'], remove=True)
'product.html?name=wired'
>>>
```

By default, URL fragments are removed. If you need to preserve fragments, pass the `keep_fragments` argument as `True`.

```
>>> w3lib.url.url_query_cleaner('http://domain.tld/?bla=123#123123', ['bla'],␣
↪remove=True, keep_fragments=True)
'http://domain.tld/#123123'
```

w3lib.url.**url_query_parameter**(*url: Union[str, bytes], parameter: str, default: Optional[str] = None, keep_blank_values: Union[bool, int] = 0*) → Optional[str]

Return the value of a url parameter, given the url and parameter name

General case:

```
>>> import w3lib.url
>>> w3lib.url.url_query_parameter("product.html?id=200&foo=bar", "id")
'200'
>>>
```

Return a default value if the parameter is not found:

```
>>> w3lib.url.url_query_parameter("product.html?id=200&foo=bar", "notthere",
↪"mydefault")
'mydefault'
>>>
```

Returns None if *keep_blank_values* not set or 0 (default):

```
>>> w3lib.url.url_query_parameter("product.html?id=", "id")
>>>
```

Returns an empty string if *keep_blank_values* set to 1:

```
>>> w3lib.url.url_query_parameter("product.html?id=", "id", keep_blank_values=1)
''
>>>
```

CHAPTER 3

Requirements

Python 3.6+

CHAPTER 4

Install

```
pip install w3lib
```

CHAPTER 5

Tests

pytest is the preferred way to run tests. Just run: `pytest` from the root directory to execute tests using the default Python interpreter.

tox could be used to run tests for all supported Python versions. Install it (using 'pip install tox') and then run `tox` from the root directory - tests will be executed for all available Python interpreters.

CHAPTER 6

Changelog

6.1 1.22.0 (2020-05-13)

- Python 3.4 is no longer supported (issue #156)
- *w3lib.url.safe_url_string()* now supports an optional `quote_path` parameter to disable the percent-encoding of the URL path (issue #119)
- *w3lib.url.add_or_replace_parameter()* and *w3lib.url.add_or_replace_parameters()* no longer remove duplicate parameters from the original query string that are not being added or replaced (issue #126)
- *w3lib.html.remove_tags()* now raises a `ValueError` exception instead of `AssertionError` when using both the `which_ones` and the `keep` parameters (issue #154)
- Test improvements (issues #143, #146, #148, #149)
- Documentation improvements (issues #140, #144, #145, #151, #152, #153)
- Code cleanup (issue #139)

6.2 1.21.0 (2019-08-09)

- Add the `encoding` and `path_encoding` parameters to *w3lib.url.safe_download_url()* (issue #118)
- *w3lib.url.safe_url_string()* now also removes tabs and new lines (issue #133)
- *w3lib.html.remove_comments()* now also removes truncated comments (issue #129)
- *w3lib.html.remove_tags_with_content()* no longer removes tags which start with the same text as one of the specified tags (issue #114)
- Recommend pytest instead of nose to run tests (issue #124)

6.3 1.20.0 (2019-01-11)

- Fix url_query_cleaner to do not append "?" to urls without a query string (issue #109)
- Add support for Python 3.7 and drop Python 3.3 (issue #113)
- Add *w3lib.url.add_or_replace_parameters* helper (issue #117)
- Documentation fixes (issue #115)

6.4 1.19.0 (2018-01-25)

- Add a workaround for CPython segfault (https://bugs.python.org/issue32583) which affect w3lib.encoding functions. This is technically **backwards incompatible** because it changes the way non-decodable bytes are replaced (in some cases instead of two `\ufffd` chars you can get one). As a side effect, the fix speeds up decoding in Python 3.4+.
- Add 'encoding' parameter for w3lib.http.basic_auth_header.
- Fix pypy testing setup, add pypy3 to CI.

6.5 1.18.0 (2017-08-03)

- Include additional assets used for distribution packages in the source tarball
- Consider `[` and `]` as safe characters in path and query components of URLs, i.e. they are not escaped anymore
- Disable codecov project coverage check

6.6 1.17.0 (2017-02-08)

- Add Python 3.5 and 3.6 support
- Add `w3lib.url.parse_data_uri` helper for parsing "data:" URIs
- Add `w3lib.html.strip_html5_whitespace` function to strip leading and trailing whitespace as per W3C recommendations, e.g. for cleaning "href" attribute values
- Fix `w3lib.http.headers_raw_to_dict` for multiple headers with same name
- Do not distribute tests/test_*.pyc artifacts

6.7 1.16.0 (2016-11-10)

- `canonicalize_url()` and `safe_url_string()`: strip ":" when no port is specified (as per RFC 3986; see also https://github.com/scrapy/scrapy/issues/2377)
- `url_query_cleaner()`: support new `keep_fragments` argument (defaulting to `False`)

6.8 1.15.0 (2016-07-29)

- Add `canonicalize_url()` to `w3lib.url`

6.9 1.14.3 (2016-07-14)

Bugfix release:

- Handle IDNA encoding failures in `safe_url_string()` (issue #62)

6.10 1.14.2 (2016-04-11)

Bugfix release:

- fix function import for (deprecated) `urljoin_rfc` (issue #51)
- only expose wanted functions from `w3lib.url`, via `__all__` (see issue #54, https://github.com/scrapy/scrapy/issues/1917)

6.11 1.14.1 (2016-04-07)

Bugfix release:

- For bytes URLs, when supplied encoding (or default UTF8) is wrong, `safe_url_string` falls back to percent-encoding offending bytes.

6.12 1.14.0 (2016-04-06)

Changes to safe_url_string:

- proper handling of non-ASCII characters in Python2 and Python3
- support IDNs
- new *path_encoding* to override default UTF-8 when serializing non-ASCII characters before percent-encoding

html_body_declared_encoding also detects encoding when not sole attribute in `<meta>`.

Package is now properly marked as `zip_safe`.

6.13 1.13.0 (2015-11-05)

- remove_tags removes uppercase tags as well;
- ignore meta-redirects inside script or noscript tags by default, but add an option to not ignore them;
- replace_entities now handles entities without trailing semicolon;
- fixed uncaught UnicodeDecodeError when decoding entities.

6.14 1.12.0 (2015-06-29)

- meta_refresh regex now handles leading newlines and whitespaces in the url;
- include tests folder in source distribution.

6.15 1.11.0 (2015-01-13)

- url_query_cleaner now supports str or list parameters;
- add support for resolving base URLs in <base> tags with attributes before href.

6.16 1.10.0 (2014-08-20)

- reverted all 1.9.0 changes.

6.17 1.9.0 (2014-08-16)

- all url-related functions accept bytes and unicode and now return bytes.

6.18 1.8.1 (2014-08-14)

- w3lib.http.basic_auth_header now returns bytes

6.19 1.8.0 (2014-07-31)

- add support for big5-hkscs encoding.

6.20 1.7.1 (2014-07-26)

- PY3 fixed headers_raw_to_dict and headers_dict_to_raw;
- documentation improvements;
- provide wheels.

6.21 1.6 (2014-06-03)

- w3lib.form.encode_multipart is deprecated;
- docstrings and docs are improved;
- w3lib.url.add_or_replace_parameter is re-implemented on top of stdlib functions;
- remove_entities is renamed to replace_entities.

6.22 1.5 (2013-11-09)

- Python 2.6 support is dropped.

6.23 1.4 (2013-10-18)

- Python 3 support;
- get_meta_refresh encoding handling is fixed;
- check for '?' in add_or_replace_parameter;
- ISO-8859-1 is used for HTTP Basic Auth;
- fixed unicode handling in replace_escape_chars;

6.24 1.3 (2012-05-13)

- support non-standard gb_2312_80 encoding;
- drop Python 2.5 support.

6.25 1.2 (2012-05-02)

- Detect encoding for content attr before http-equiv in meta tag.

6.26 1.1 (2012-04-18)

- w3lib.html.remove_comments handles multiline comments;
- Added w3lib.encoding module, containing functions for working with character encoding, like encoding autodetection from HTML pages.
- w3lib.url.urljoin_rfc is deprecated.

6.27 1.0 (2011-04-17)

First release of w3lib.

6.28 History

The code of w3lib was originally part of the Scrapy framework but was later stripped out of Scrapy, with the aim of make it more reusable and to provide a useful library of web functions without depending on Scrapy.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

W

Index