20.16. urlparse — Parse URLs into components

Note: The urlparse module is renamed to urllib.parse in Python 3.0. The 2to3 tool will automatically adapt imports when converting your sources to 3.0.

Source code: Lib/urlparse.py

This module defines a standard interface to break Uniform Resource Locator (URL) strings up in components (addressing scheme, network location, path etc.), to combine the components back into a URL string, and to convert a "relative URL" to an absolute URL given a "base URL."

The module has been designed to match the Internet RFC on Relative Uniform Resource Locators (and discovered a bug in an earlier draft!). It supports the following URL schemes: file, ftp, gopher, hdl, http, https, imap, mailto, mms, news, nntp, prospero, rsync, rtsp, rtspu, sftp, shttp, sip, sips, snews, svn, svn+ssh, telnet, wais.

New in version 2.5: Support for the sftp and sips schemes.

The urlparse module defines the following functions:

urlparse.urlparse(urlstring[, scheme[, allow_fragments]])
Parse a URL into six components, returning a 6-tuple. This corresponds to the general structure of a URL: `scheme://netloc/path;parameters?query#fragment`. Each tuple item is a string, possibly empty. The components are not broken up in smaller parts (for example, the network location is a single string), and % escapes are not expanded. The delimiters as shown above are not part of the result, except for a leading slash in the path component, which is retained if present. For example:

```python
>>> from urlparse import urlparse
>>> o = urlparse('http://www.cwi.nl:80/%7Eguido/Python.html')
>>> o
ParseResult(scheme='http', netloc='www.cwi.nl:80', path='/%7Eguido/', params='', query='', fragment='')
>>> o.scheme
'http'
>>> o.port
80
>>> o.geturl()
'http://www.cwi.nl:80/%7Eguido/Python.html'
```

Following the syntax specifications in RFC 1808, urlparse recognizes a netloc only if it is properly introduced by '//'. Otherwise the input is presumed to be a relative URL and thus to start with a path component.

```python
>>> from urlparse import urlparse
```
If the `scheme` argument is specified, it gives the default addressing scheme, to be used only if the URL does not specify one. The default value for this argument is the empty string.

If the `allow_fragments` argument is false, fragment identifiers are not allowed, even if the URL’s addressing scheme normally does support them. The default value for this argument is `True`.

The return value is actually an instance of a subclass of `tuple`. This class has the following additional read-only convenience attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Index</th>
<th>Value</th>
<th>Value if not present</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme</td>
<td>0</td>
<td>URL scheme specifier</td>
<td>empty string</td>
</tr>
<tr>
<td>netloc</td>
<td>1</td>
<td>Network location part</td>
<td>empty string</td>
</tr>
<tr>
<td>path</td>
<td>2</td>
<td>Hierarchical path</td>
<td>empty string</td>
</tr>
<tr>
<td>params</td>
<td>3</td>
<td>Parameters for last path element</td>
<td>empty string</td>
</tr>
<tr>
<td>query</td>
<td>4</td>
<td>Query component</td>
<td>empty string</td>
</tr>
<tr>
<td>fragment</td>
<td>5</td>
<td>Fragment identifier</td>
<td>empty string</td>
</tr>
<tr>
<td>username</td>
<td></td>
<td>User name</td>
<td>None</td>
</tr>
<tr>
<td>password</td>
<td></td>
<td>Password</td>
<td>None</td>
</tr>
<tr>
<td>hostname</td>
<td></td>
<td>Host name (lower case)</td>
<td>None</td>
</tr>
<tr>
<td>port</td>
<td></td>
<td>Port number as integer, if present</td>
<td>None</td>
</tr>
</tbody>
</table>

See section `Results of urlparse()` and `urlsplit()` for more information on the result object.

*Changed in version 2.5:* Added attributes to return value.

*Changed in version 2.7:* Added IPv6 URL parsing capabilities.

`urlparse.parse_qs(qs[, keep_blank_values[, strict_parsing]])`

Parse a query string given as a string argument (data of type `application/x-www-form-urlencoded`). Data are returned as a dictionary. The dictionary keys are the unique query variable names and the values are lists of values for each name.

The optional argument `keep_blank_values` is a flag indicating whether blank
values in percent-encoded queries should be treated as blank strings. A true value indicates that blanks should be retained as blank strings. The default false value indicates that blank values are to be ignored and treated as if they were not included.

The optional argument `strict_parsing` is a flag indicating what to do with parsing errors. If false (the default), errors are silently ignored. If true, errors raise a `ValueError` exception.

Use the `urllib.urlencode()` function to convert such dictionaries into query strings.

*New in version 2.6:* Copied from the `cgi` module.

```python
urlparse.parse_qsl(qs[, keep_blank_values[, strict_parsing]])
```

Parse a query string given as a string argument (data of type `application/x-www-form-urlencoded`). Data are returned as a list of name, value pairs.

The optional argument `keep_blank_values` is a flag indicating whether blank values in percent-encoded queries should be treated as blank strings. A true value indicates that blanks should be retained as blank strings. The default false value indicates that blank values are to be ignored and treated as if they were not included.

The optional argument `strict_parsing` is a flag indicating what to do with parsing errors. If false (the default), errors are silently ignored. If true, errors raise a `ValueError` exception.

Use the `urllib.urlencode()` function to convert such lists of pairs into query strings.

*New in version 2.6:* Copied from the `cgi` module.

```python
urlparse.urlunparse(parts)
```

Construct a URL from a tuple as returned by `urlparse()`. The `parts` argument can be any six-item iterable. This may result in a slightly different, but equivalent URL, if the URL that was parsed originally had unnecessary delimiters (for example, a `?` with an empty query; the RFC states that these are equivalent).

```python
urlparse.ursplit(urlstring[, scheme[, allow_fragments]])
```

This is similar to `urlparse()`, but does not split the params from the URL. This should generally be used instead of `urlparse()` if the more recent URL syntax allowing parameters to be applied to each segment of the `path` portion of the URL (see RFC 2396) is wanted. A separate function is needed to separate the path segments and parameters. This function returns a 5-tuple: (addressing scheme, network location, path, query, fragment identifier).

The return value is actually an instance of a subclass of `tuple`. This class has the following additional read-only convenience attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Index</th>
<th>Value</th>
<th>Value if not present</th>
</tr>
</thead>
</table>

3 of 6 05/09/2012 03:53 PM
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Index</th>
<th>Value</th>
<th>Value if not present</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme</td>
<td>0</td>
<td>URL scheme specifier</td>
<td>empty string</td>
</tr>
<tr>
<td>netloc</td>
<td>1</td>
<td>Network location part</td>
<td>empty string</td>
</tr>
<tr>
<td>path</td>
<td>2</td>
<td>Hierarchical path</td>
<td>empty string</td>
</tr>
<tr>
<td>query</td>
<td>3</td>
<td>Query component</td>
<td>empty string</td>
</tr>
<tr>
<td>fragment</td>
<td>4</td>
<td>Fragment identifier</td>
<td>empty string</td>
</tr>
<tr>
<td>username</td>
<td></td>
<td>User name</td>
<td>None</td>
</tr>
<tr>
<td>password</td>
<td></td>
<td>Password</td>
<td>None</td>
</tr>
<tr>
<td>hostname</td>
<td></td>
<td>Host name (lower case)</td>
<td>None</td>
</tr>
<tr>
<td>port</td>
<td></td>
<td>Port number as integer, if present</td>
<td>None</td>
</tr>
</tbody>
</table>

See section *Results of urlparse() and urlsplit()* for more information on the result object.

*New in version 2.2.*

*Changed in version 2.5:* Added attributes to return value.

**urlparse. urlunsplit(parts)**

Combine the elements of a tuple as returned by `urlsplit()` into a complete URL as a string. The parts argument can be any five-item iterable. This may result in a slightly different, but equivalent URL, if the URL that was parsed originally had unnecessary delimiters (for example, a ? with an empty query; the RFC states that these are equivalent).

*New in version 2.2.*

**urlparse. urljoin(base, url[, allow_fragments])**

Construct a full (“absolute”) URL by combining a “base URL” (base) with another URL (url). Informally, this uses components of the base URL, in particular the addressing scheme, the network location and (part of) the path, to provide missing components in the relative URL. For example:

```python
>>> from urlparse import urljoin
>>> urljoin('http://www.cwi.nl/%7Eguido/Python.html', 'FAQ.html')
'http://www.cwi.nl/%7Eguido/FAQ.html'
```

The `allow_fragments` argument has the same meaning and default as for `urlparse()`.

**Note:** If `url` is an absolute URL (that is, starting with // or scheme://), the `url`'s host name and/or scheme will be present in the result. For example:

```python
>>> urljoin('http://www.cwi.nl/%7Eguido/Python.html',
          'http://www.python.org/%7Eguido')
'http://www.python.org/%7Eguido'
```
If you do not want that behavior, preprocess the url with `urlsplit()` and `urlunsplit()`, removing possible `scheme` and `netloc` parts.

`urlparse.urldefrag(url)`

If `url` contains a fragment identifier, returns a modified version of `url` with no fragment identifier, and the fragment identifier as a separate string. If there is no fragment identifier in `url`, returns `url` unmodified and an empty string.

See also:

- **RFC 3986 - Uniform Resource Identifiers**
  This is the current standard (STD66). Any changes to urlparse module should conform to this. Certain deviations could be observed, which are mostly due to backward compatibility purposes and for certain de-facto parsing requirements as commonly observed in major browsers.

- **RFC 2732 - Format for Literal IPv6 Addresses in URL's.**
  This specifies the parsing requirements of IPv6 URLs.

- **RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax**
  Document describing the generic syntactic requirements for both Uniform Resource Names (URNs) and Uniform Resource Locators (URLs).

- **RFC 2368 - The mailto URL scheme.**
  Parsing requirements for mailto url schemes.

- **RFC 1808 - Relative Uniform Resource Locators**
  This Request For Comments includes the rules for joining an absolute and a relative URL, including a fair number of “Abnormal Examples” which govern the treatment of border cases.

- **RFC 1738 - Uniform Resource Locators (URL)**
  This specifies the formal syntax and semantics of absolute URLs.

### 20.16.1. Results of `urlparse()` and `urlsplit()`

The result objects from the `urlparse()` and `urlsplit()` functions are subclasses of the `tuple` type. These subclasses add the attributes described in those functions, as well as provide an additional method:

`ParseResult.geturl()`

Return the re-combined version of the original URL as a string. This may differ from the original URL in that the scheme will always be normalized to lower case and empty components may be dropped. Specifically, empty parameters, queries, and fragment identifiers will be removed.

The result of this method is a fixpoint if passed back through the original parsing function:

```python
>>> import urlparse
>>> url = 'HTTP://www.Python.org/doc/#'
```
New in version 2.5.

The following classes provide the implementations of the parse results:

**class urlparse.BaseResult**

Base class for the concrete result classes. This provides most of the attribute definitions. It does not provide a geturl() method. It is derived from tuple, but does not override the __init__() or __new__() methods.

**class urlparse.ParseResult**(scheme, netloc, path, params, query, fragment)

Concrete class for urlparse() results. The __new__() method is overridden to support checking that the right number of arguments are passed.

**class urlparse.SplitResult**(scheme, netloc, path, query, fragment)

Concrete class for urlsplit() results. The __new__() method is overridden to support checking that the right number of arguments are passed.

```python
>>> r1 = urlparse.urlsplit(url)
>>> r1.geturl()
'http://www.Python.org/doc/

>>> r2 = urlparse.urlsplit(r1.geturl())
>>> r2.geturl()
'http://www.Python.org/doc/
```