eGenix.com

mxURL

Flexible URL Data-Type for Python

Version 3.2

Copyright © 2001-2013 by eGenix.com GmbH, Langenfeld

All rights reserved. No part of this work may be reproduced or used in a any form or by any means without written permission of the publisher.

All product names and logos are trademarks of their respective owners.

The product names "mxBeeBase", "mxCGIPython", "mxCounter", "mxCrypto", "mxDateTime", "mxHTMLTools", "mxIP", "mxLicenseManager", "mxLog", "mxNumber", "mxODBC", "mxODBC Connect", "mxODBC Zope DA", "mxObjectStore", "mxProxy", "mxQueue", "mxStack", "mxTextTools", "mxTidy", "mxTools", "mxUID", "mxURL", "mxXMLTools", "eGenix Application Server", "PythonHTML", "eGenix" and "eGenix.com" and corresponding logos are trademarks or registered trademarks of eGenix.com GmbH, Langenfeld

Printed in Germany.

Contents

1. Introduction

This package provides a new datatype for storing and manipulating URL values as well as a few helpers related to URL building, encoding and decoding.

The main intention of the package is to provide an easy to use, fast and lightweight datatype for Universal Resource Locators. The W3C now calls these URIs -- Universal Resource Identifiers.

2. mx.URL.URL Object

To simplify and speed up handling URLs the package provides a new type to work with them in an object oriented way.

URL objects can be added to each other as well as right added to strings giving a joined URL object in both cases. The join semantics depend on the URL schemes and their attributes.

2.1 URL Parts

The URL parts are defined as follows:

```
scheme://user:passwd@host:port/path;params?query#fragment
```

Whether or not certain parts of the URL are required depends on the scheme.

`scheme`

URL scheme defining the network resource type.

`user`

User name

`passwd`

Password

`host`

Host name

`port`

Port number.

Schemes usually define a default port, so providing a port is usually not necessary.

`path`

Path to the resource using the slash ('/') as path component separator.

`params`

Optional parameters.

These are rarely used and not always supported by all applications dealing with URLs.

`query`

Query part, usually in the format key=value with key-value-pairs separated by ampersands (&).

`fragment`

If a URL refers to part of a document, the fragment indicates the name of the part.

Note that we abbreviate the network location part (`//user:passwd@host:port/`) as *netloc*.

2.2 Predefined URL schemes

These schemes are predefined by the module. The function `register_scheme()` (see above) allows adding new ones or changing the behavior for predefined ones.

Scheme	*uses_netloc*	*uses_params*	*uses_query*	*uses_fragment*	*uses_relative*
http	1	1	1	1	1
https	1	1	1	1	1
shttp	1	1	1	1	1
mailto	0	0	1	0	0
ftp	1	1	0	1	1
gopher	1	0	0	1	1
news	1	0	0	1	1
nntp	1	0	0	0	1
telnet	1	0	0	1	0

Scheme	*uses_netloc*	*uses_params*	*uses_query*	*uses_fragment*	*uses_relative*
file	1	0	0	0	1
about	0	0	0	0	0
javascript	0	0	0	0	0
ldap	1	0	0	0	0
svn+ssh	1	0	0	0	1

The `uses_*` fields are integers 0 or 1 representing the schemes possibilities. When a feature is set to 0 the corresponding field is left out while parsing the URL. Characters which would normally be seen as separators are ignored then.

`uses_relative` is important when joining URLs. Only URLs with `uses_relative` will have their paths joined according to the common rules.

Note that the URL object constructors will raise a `ValueError` exception for unknown schemes they find in the construction string.

2.3 Joining URLs

One of the most common operations when dealing with URLs is to join a URL part to a base URL.

In mxURL, this operation is triggered by simply adding two URL objects, or simply adding a string to a left-hand URL object. The result will be a new URL object.

mxURL tries to follow the standard RFC 3986 in most detail, but doesn't implement all cases.[1]

[1] See the included mx/URL/mxURL/test.py for details. We will gradually update mxURL to fully support the RFCs in upcoming versions. If you find important features missing, please report them.

Examples

```
>>> from mx.URL import URL
>>> URL('http://egenix.com/') + URL('products/python')
<URL:http://egenix.com/products/python>
>>> URL('http://egenix.com/') + URL('products/python/mxBase')
<URL:http://egenix.com/products/python/mxBase>
>>> URL('http://egenix.com/products/python') +
URL('../../services')            <URL:http://egenix.com/services>
>>> URL('http://egenix.com/products/python') +
URL('../../../services')
<URL:http://egenix.com/services>
>>> URL('http://egenix.com/products/python') +
URL('https://egenix.com')        <URL:https://egenix.com>
>>> URL('/products/python') + URL('../../services')
<URL:/services>
>>> URL('/products/python') + URL('../services')
<URL:/services>
>>> URL('a') + URL('b')
<URL:b>
>>> URL('a/') + URL('b')
<URL:a/b>
>>> URL('a/') + URL('b/')
<URL:a/b/>
>>> URL('/a/') + URL('b/')
<URL:/a/b/>
>>> URL('/a') + URL('b/')
<URL:/b/>
```

2.4 Using URL objects with strings

URL objects accept addition with strings when used on the left-hand side. mxURL will first convert the string to an URL object and then apply the operation.

It is also possible to convert URL objects back to strings using the `str()` constructor or the `%s` formatting parameter. Conversion to Unicode is also possible, using the `unicode()` or `%s` formatting parameter in a Unicode string.

Interaction between URL objects and strings:

```
>>> URL('http://egenix.com/products/python') +
URL('../../services')
<URL:http://egenix.com/services>
>>> URL('http://egenix.com/products/python') + '../../services'
<URL:http://egenix.com/services>
>>> u = URL('http://egenix.com/products/python') + '../../services'
>>> str(u)
'http://egenix.com/services'
```

Using strings on the left-hand side of the addition does not work[2]:

```
>>> 'http://egenix.com/products/python' + URL('../../services')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'URL' objects
```

2.5 mx.URL.URL Object Constructors

These constructors are available in the package:

`URL(url)`

Create a new URL object from url. Takes either a string or another URL as argument. The url is stored normalized.

`RawURL(url)`

Create a new URL object from url. Takes either a string or another URL as argument. The url is not normalized but stored as-is.

`BuildURL(scheme='', netloc='', path='', params='', query='', fragment='')`

Create a new URL object from the given parameters. The url is stored normalized. This constructor can handle keywords.

Normalizing means that unnecessary relative components and slashes are removed from the URL prior to storing it. The stored URL will always be equivalent to the one given to the constructor.

> The URL type uses a scheme feature dictionary to figure out how to parse different schemes. Use the `mx.URL.add_scheme()` function to access this dictionary.

2.6 mx.URL.URL Object Instance Variables

A `URL` instance `url` provides access to these (read-only) variables:

`.absolute`

1 iff the object's path is absolute; 0 otherwise.

[2] We may add this functionality in a future version of mxURL.

`.base`

Base part of the URL's path: everything excluding a possibly given file with extension.

`.ext`

Extension (without dot) of the file given in the URL converted to lowercase letters.

The extension is defined as the last dotted part of the filename. In some cases, this may clash with the way the OS uses filenames, e.g. on Unix it is common to hide files by using a dot as first character of the filename.

`.file`

File name of the file pointed to by the URL. Directories are not included.

`.fragment`

Fragment part of the URL without the '#'.

`.host`

Hostname included in the network location part of the URL (//user:passwd@host:port/) or ''.

`.mimetype`

The MIME type as string ("major/minor") or "*/*" if it cannot be determined. The package uses the `types_map` dictionary of the Python standard lib's `mimetype` module as basis for finding out the MIME type. You can add entries to that dictionary at runtime to adapt the mechanism to your needs.

`.netloc`

Network location as given in the URL without the leading '//' and possibly terminating '/'. Username and password are included if given (//user:passwd@host:port/).

`.normal`

1 iff the object's path has been normalized; 0 otherwise.

`.params`

Parameter section of the URL without the ';'.

`.passwd`

Password included in the network location part of the URL (//user:passwd@host:port/) or ''.

`.path`

Path as given in the URL. If the URL contains a netloc part this will always start with a '/'.

`.port`

Port included in the network location part of the URL (//user:passwd@host:port/) as integer.

`.scheme`

Access scheme without the terminating ':'.

`.string`

The complete URL as string.

`.url`

The complete URL as string. Alias for `.string`.

`.user`

Username included in the network location part of the URL (//user:passwd@host:port/) or ''.

2.7 mx.URL.URL Object Instance Methods

A `URL` object `url` defines these methods:

`.basic()`

Return a new URL object pointing to the same base URL, but without the parts params, query and fragment.

In case the url already fulfills this requirement, a new reference to it is returned.

`.depth()`

Return the depth of the URL.

Depth is only defined if the URL is normalized and absolute. If the URL is not absolute or contains relative components (e.g. /a/../b/) an error will be raised.

The top level has depth 0.

`.normalized()`

Return a new URL object pointing to the same URL but normalized.

`.parsed()`

Return a tuple `(scheme, netloc, path, params, query, fragment)` just as `urlparse.urlparse()` does.

`.pathentry(index)`

Return the path entry index.

index may be negative to indicate an entry counted from the right (with -1 being the rightmost entry). An `IndexError` is raised in case the index lies out of range. Leading and trailing slashes are not counted.

`.pathlen()`

Return the path length as defined by the `.pathentry()` method.

Leading and trailing slashes are not counted.

`.pathtuple()`

Return the path as tuple of strings.

Leading and trailing slashes are ignored and the slashes are not included.

`.relative(baseURL)`

Return a new URL object that when joined with `baseURL` results in the same URL as the object itself.

URL and `baseURL` must both be absolute URLs for this to work. An exception is raised otherwise.

The `baseURL` should provide scheme and netloc, because otherwise joining might result in loss of scheme information. If only the URL provides a scheme, then the returned relative URL will also include that scheme.

Parameters, fragment and query of the URL object are preserved; only the path is made relative and the netloc removed (relative paths and netlocs don't go together).

In case both URLs provide schemes and/or netlocs that point to different resources, the method simply returns a new reference to the object.

`.rebuild(scheme='', netloc='', path='', params='', query='', fragment='')`

Return a new URL object created from the given parameters and the URL object. This method can handle keywords.

Arguments not given are taken unchanged from the URL object.

3. mx.URL Functions

The mxURL package defines the following functions:

These functions are available.

`addscheme(url)`

Returns the URL url with scheme added according to common usage.

If the url already provides a scheme, nothing is changed. Strings are turned into URL object by the function.

These conventions are used:

- www. -> http://www.
- ftp. -> ftp://ftp.
- [/.] -> file:[/.]
- none of these -> http://

`escape(url)`

Escape all special chars in a URL using %xx hex encodings.

`queryencode(items, prefix='?')`

Takes a sequence of key,value items and formats a URL encoded query part out of it.

Keys and values are converted to string prior to URL conversion.

prefix is prepended to the resulting string. It defaults to '?' so that the returned value can directly be concatenated to a URL.

`querydecode(query)`

Decodes a query string and returns a list of items (key, value).

If query is prefixed with a question mark ('?'), this is silently ignored.

Query parts which don't provide a value will get None assigned as value in the items list.

`quote(url)`

Alias for escape().

`register_mimetype(extension,major='*', minor='*')`

Adds a new mime type to the registry used by mxURL.

extension must be a file name extension including the delimiting dot (e.g. ".html"). The function will overwrite any existing entry for the given extension.

Note that mxURL uses the dictionary `mimetypes.types_map` as its registry, so changes done with this function will be available through the `mimetypes` module too.

`register_scheme(scheme, uses_netloc, uses_params, uses_query, uses_fragment, uses_relative)`

Adds a new scheme to the URL objects scheme registry. See below for an explanation of the parameters.

`unescape(url)`

Unescape a URL containing %xx-character encodings.

`unquote(url)`

Alias for unescape().

`urlencode(url)`

Alias for escape().

`urldecode(url)`

Alias for unescape().

`urljoin(u,v)`

Takes two URLs or strings, joins them and returns the result as URL object (see below).

4. mx.URL Constants

`Error`

Error class used for package specific errors. It is a subclass of `StandardError`.

5. Examples of Use

Here is a very simple one:

```
from mx.URL import *
url = URL('http://search.python.org/query.html?qt=mx')
print url.scheme
>>> http
print url.host
>>> search.python.org
print url.query
>>> qt=mx
print url.path
>>> /query.html
```

More examples will eventually appear in the Examples/ subdirectory of the package.

6. Package Structure

```
[URL]
        Doc/
        [mxURL]
        URL.py
```

Entries enclosed in brackets are packages (i.e. they are directories that include a `__init__.py` file). Ones without brackets are just simple subdirectories that are not accessible via `import`.

The package imports all symbols from the extension module, so you only need to '`from mx import URL`' to start working.

7. Support

eGenix.com is providing commercial support for this package. If you are interested in receiving information about this service please see the *eGenix.com Support Conditions*.

8. Copyright & License

© 2001-2013, Copyright by eGenix.com Software GmbH, Langenfeld, Germany; All Rights Reserved. mailto: *info@egenix.com*

This software is covered by the ***eGenix.com Public License Agreement***, which is included in the following section. The text of the license is also included as file "LICENSE" in the package's main directory.

By downloading, copying, installing or otherwise using the software, you agree to be bound by the terms and conditions of the following *eGenix.com Public License Agreement*.

EGENIX.COM PUBLIC LICENSE AGREEMENT

Version 1.1.0

This license agreement is based on the Python CNRI License Agreement, a widely accepted open-source license.

1. Introduction

This "License Agreement" is between eGenix.com Software, Skills and Services GmbH ("eGenix.com"), having an office at Pastor-Loeh-Str. 48, D-40764 Langenfeld, Germany, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. License

Subject to the terms and conditions of this eGenix.com Public License Agreement, eGenix.com hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the eGenix.com Public License Agreement is retained in the Software, or in any derivative version of the Software prepared by Licensee.

3. NO WARRANTY

eGenix.com is making the Software available to Licensee on an "AS IS" basis. SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, EGENIX.COM MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, EGENIX.COM MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. LIMITATION OF LIABILITY

EGENIX.COM SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) AS A RESULT OF USING, MODIFYING OR

DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO LICENSEE.

5. Termination

This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. Third Party Rights

Any software or documentation in source or binary form provided along with the Software that is associated with a separate license agreement is licensed to Licensee under the terms of that license agreement. This License Agreement does not apply to those portions of the Software. Copies of the third party licenses are included in the Software Distribution.

7. General

Nothing in this License Agreement affects any statutory rights of consumers that cannot be waived or limited by contract.

Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between eGenix.com and Licensee.

If any provision of this License Agreement shall be unlawful, void, or for any reason unenforceable, such provision shall be modified to the extent necessary to render it enforceable without losing its intent, or, if no such modification is possible, be severed from this License Agreement and shall not affect the validity and enforceability of the remaining provisions of this License Agreement.

This License Agreement shall be governed by and interpreted in all respects by the law of Germany, excluding conflict of law provisions. It shall not be governed by the United Nations Convention on Contracts for International Sale of Goods.

This License Agreement does not grant permission to use eGenix.com trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party.

The controlling language of this License Agreement is English. If Licensee has received a translation into another language, it has been provided for Licensee's convenience only.

8. Agreement

By downloading, copying, installing or otherwise using the Software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

For question regarding this License Agreement, please write to:

eGenix.com Software, Skills and Services GmbH

Pastor-Loeh-Str. 48

D-40764 Langenfeld

Germany