`!email.utils`: Miscellaneous utilities

email.utils

Source code: `Lib/email/utils.py`

There are a couple of useful utilities provided in the `!email.utils` module:

localtime(dt=None)

Return local time as an aware datetime object. If called without arguments, return the current time. Otherwise the *dt* argument should be a `~datetime.datetime` instance, and it is converted to the local time zone according to the system time zone database. If *dt* is naive (that is, `dt.tzinfo` is `None`), it is assumed to be in local time.

3.3

3.12 3.14 The *isdst* parameter.

make_msgid(idstring=None, domain=None)

Returns a string suitable for an `2822`-compliant `Message-ID` header. Optional *idstring*, if given, is a string used to strengthen the uniqueness of the message id. Optional *domain*, if given, provides the portion of the msgid after the '@'. The default is the local hostname. It is normally not necessary to override this default, but may be useful in certain cases, such as constructing a distributed system that uses a consistent domain name across multiple hosts.

3.2 Added the *domain* keyword.

The remaining functions are part of the legacy (`Compat32`) email API. There is no need to directly use these with the new API, since the parsing and formatting they provide is done automatically by the header parsing machinery of the new API.

quote(str)

Return a new string with backslashes in *str* replaced by two backslashes, and double quotes replaced by backslash-double quote.

unquote(str)

Return a new string which is an *unquoted* version of *str*. If *str* ends and begins with double quotes, they are stripped off. Likewise if *str* ends and begins with angle brackets, they are stripped off.

parseaddr(address, *, strict=True)

Parse address -- which should be the value of some address-containing field such as `To` or `Cc` -- into its constituent *realname* and *email address* parts. Returns a tuple of that information, unless the parse fails, in which case a 2-tuple of `('', '')` is returned.

If *strict* is true, use a strict parser which rejects malformed inputs.

3.13 Add *strict* optional parameter and reject malformed inputs by default.

formataddr(pair, charset='utf-8', *, strict=True)

The inverse of `parseaddr`, this takes a 2-tuple of the form `(realname, email_address)` and returns the string value suitable for a `To` or `Cc` header. If the first element of *pair* is false, then the second element is returned unmodified.

Optional *charset* is the character set that will be used in the `2047` encoding of the `realname` if the `realname` contains non-ASCII characters. Can be an instance of `str` or a `~email.charset.Charset`. Defaults to `utf-8`.

If *strict* is true (the default), raise `ValueError` for inputs that contain CR or LF, which are not allowed in an email address. Set *strict* to `False` to allow non-strict inputs.

3.3 Added the *charset* option.

3.16 Added the *strict* parameter.

getaddresses(fieldvalues, *, strict=True)

This function returns a list of 2-tuples of the form returned by `parseaddr`. *fieldvalues* is a sequence of header field values as might be returned by `Message.get_all() <email.message.Message.get_all>`.

If *strict* is true, use a strict parser which rejects malformed inputs.

Here's a simple example that gets all the recipients of a message:

```
from email.utils import getaddresses


tos = msg.get_all('to', [])
ccs = msg.get_all('cc', [])
```

```
resent_tos = msg.get_all('resent-to', [])
resent_ccs = msg.get_all('resent-cc', [])
all_recipients = getaddresses(tos + ccs + resent_tos + resent_ccs)
```

3.13 Add *strict* optional parameter and reject malformed inputs by default.

parsedate(date)

Attempts to parse a date according to the rules in `2822#section-3.3`. However, some mailers don't follow that format as specified, so `parsedate` tries to guess correctly in such cases. *date* is a string containing an `2822` date, such as `"Mon, 20 Nov 1995 19:12:08 -0500"`. If it succeeds in parsing the date, `parsedate` returns a 9-tuple that can be passed directly to `time.mktime`; otherwise `None` will be returned. Note that indexes 6, 7, and 8 of the result tuple are not usable.

parsedate_tz(date)

Performs the same function as `parsedate`, but returns either `None` or a 10-tuple; the first 9 elements make up a tuple that can be passed directly to `time.mktime`, and the tenth is the offset of the date's timezone from UTC (which is the official term for Greenwich Mean Time)[^1]. If the input string has no timezone, the last element of the tuple returned is `0`, which represents UTC. Note that indexes 6, 7, and 8 of the result tuple are not usable.

parsedate_to_datetime(date)

The inverse of `format_datetime`. Performs the same function as `parsedate`, but on success returns a `~datetime.datetime`; otherwise `ValueError` is raised if *date* contains an invalid value such as an hour greater than 23 or a timezone offset not between -24 and 24 hours. If the input date has a timezone of `-0000`, the `datetime` will be a naive `datetime`, and if the date conforms to the RFCs it will represent a time in UTC but with no indication of the actual source timezone of the message the date comes from. If the input date has any other valid timezone offset, the `datetime` will be an aware `datetime` whose corresponding `~datetime.datetime.tzinfo` attribute is a `~datetime.timezone` object.

3.3

mktime_tz(tuple)

Turn a 10-tuple as returned by `parsedate_tz` into a UTC timestamp (seconds since the Epoch). If the timezone item in the tuple is `None`, assume local time.

formatdate(timeval=None, localtime=False, usegmt=False)

Returns a date string as per `2822`, for example:

```
Fri, 09 Nov 2001 01:08:47 -0000
```

Optional *timeval*, if given, is a floating-point time value as accepted by `time.gmtime` and `time.localtime`, otherwise the current time is used.

Optional *localtime* is a flag that when `True`, interprets *timeval*, and returns a date relative to the local timezone instead of UTC, properly taking daylight saving time into account. The default is `False` meaning UTC is used.

Optional *usegmt* is a flag that when `True`, outputs a date string with the timezone as an ASCII string `GMT`, rather than a numeric `-0000`. This is needed for some protocols (such as HTTP). This only applies when *localtime* is `False`. The default is `False`.

format_datetime(dt, usegmt=False)

Like `formatdate`, but the input is a `~datetime.datetime` instance. If it is a naive datetime, it is assumed to be "UTC with no information about the source timezone", and the conventional `-0000` is used for the timezone. If it is an aware `datetime`, then the numeric timezone offset is used. If it is an aware timezone with offset zero, then *usegmt* may be set to `True`, in which case the string `GMT` is used instead of the numeric timezone offset. This provides a way to generate standards-conformant HTTP date headers.

3.3

decode_rfc2231(s)

Decode the string *s* according to `2231`.

encode_rfc2231(s, charset=None, language=None)

Encode the string *s* according to `2231`. Optional *charset* and *language*, if given, are the character set name and language name to use. If neither is given, *s* is returned as-is. If *charset* is given but *language* is not, the string is encoded using the empty string for *language*.

collapse_rfc2231_value(value, errors='replace', fallback_charset='us-ascii')

When a header parameter is encoded in `2231` format, `Message.get_param() <email.message.Message.get_param>` may return a 3-tuple containing the character set, language, and value. `collapse_rfc2231_value` turns this into a string. Optional *errors* is passed to the *errors* argument of `str`'s `~str.encode` method; it defaults to `'replace'`. Optional *fallback_charset* specifies the character set to use if the one in the `2231` header is not known by Python; it defaults to `'us-ascii'`.

For convenience, if the *value* passed to `collapse_rfc2231_value` is not a tuple, it should be a string and it is returned unquoted.

decode_params(params)

Decode parameters list according to `2231`. *params* is a sequence of 2-tuples containing elements of the form `(content-type, string-value)`.

Footnotes

[^1]: Note that the sign of the timezone offset is the opposite of the sign of the `time.timezone` variable for the same timezone; the latter variable follows the POSIX standard while this module follows `2822`.

□□□ #2
□ □□□ □□□ 2026-10-31 17:09:46 UTC
□ □□□ □□□ 2026-10-31 17:09:47 UTC