19.1. `email` — An email and MIME handling package¶

New in version 2.2.

The `email` package is a library for managing email messages, including MIME and other **RFC 2822**-based message documents. It subsumes most of the functionality in several older standard modules such as `rfc822`, `mimetools`, `multifile`, and other non-standard packages such as `mimecntl`. It is specifically *not* designed to do any sending of email messages to SMTP (**RFC 2821**), NNTP, or other servers; those are functions of modules such as `smtplib` and `nntplib`. The `email` package attempts to be as RFC-compliant as possible, supporting in addition to **RFC 2822**, such MIME-related RFCs as **RFC 2045**, **RFC 2046**, **RFC 2047**, and **RFC 2231**.

The primary distinguishing feature of the `email` package is that it splits the parsing and generating of email messages from the internal *object model* representation of email. Applications using the `email` package deal primarily with objects; you can add sub-objects to messages, remove sub-objects from messages, completely re-arrange the contents, etc. There is a separate parser and a separate generator which handles the transformation from flat text to the object model, and then back to flat text again. There are also handy subclasses for some common MIME object types, and a few miscellaneous utilities that help with such common tasks as extracting and parsing message field values, creating RFC-compliant dates, etc.

The following sections describe the functionality of the `email` package. The ordering follows a progression that should be common in applications: an email message is read as flat text from a file or other source, the text is parsed to produce the object structure of the email message, this structure is manipulated, and finally, the object tree is rendered back into flat text.

It is perfectly feasible to create the object structure out of whole cloth — i.e. completely from scratch. From there, a similar progression can be taken as above.

Also included are detailed specifications of all the classes and modules that the `email` package provides, the exception classes you might encounter while using the `email` package, some auxiliary utilities, and a few examples. For users of the older `mimelib` package, or previous versions of the `email` package, a section on differences and porting is provided.

Contents of the `email` package documentation:

See also

Module `smtplib`
SMTP protocol client
Module `nntplib`
NNTP protocol client

19.1.12. Package History¶

This table describes the release history of the email package, corresponding to the version of Python that the package was released with. For purposes of this document, when you see a note about change or added versions, these refer to the Python version the change was made in, *not* the email package version. This table also describes the Python compatibility of each version of the package.

email version	distributed with	compatible with
`1.x`	Python 2.2.0 to Python 2.2.1	*no longer supported*
`2.5`	Python 2.2.2+ and Python 2.3	Python 2.1 to 2.5
`3.0`	Python 2.4	Python 2.3 to 2.5
`4.0`	Python 2.5	Python 2.3 to 2.5

Here are the major differences between `email` version 4 and version 3:

All modules have been renamed according to **PEP 8** standards. For example, the version 3 module `email.Message` was renamed to `email.message` in version 4.

A new subpackage `email.mime` was added and all the version 3 `email.MIME*` modules were renamed and situated into the `email.mime` subpackage. For example, the version 3 module `email.MIMEText` was renamed to `email.mime.text`.

Note that the version 3 names will continue to work until Python 2.6.

The `email.mime.application` module was added, which contains the `MIMEApplication` class.

Methods that were deprecated in version 3 have been removed. These include `Generator.__call__()`, `Message.get_type()`, `Message.get_main_type()`, `Message.get_subtype()`.

Fixes have been added for **RFC 2231** support which can change some of the return types for `Message.get_param()` and friends. Under some circumstances, values which used to return a 3-tuple now return simple strings (specifically, if all extended parameter segments were unencoded, there is no language and charset designation expected, so the return type is now a simple string). Also, %-decoding used to be done for both encoded and unencoded segments; this decoding is now done only for encoded segments.

Here are the major differences between `email` version 3 and version 2:

- The `FeedParser` class was introduced, and the `Parser` class was implemented in terms of the `FeedParser`. All parsing therefore is non-strict, and parsing will make a best effort never to raise an exception. Problems found while parsing messages are stored in the message's *defect* attribute.
- All aspects of the API which raised `DeprecationWarning`s in version 2 have been removed. These include the *_encoder* argument to the `MIMEText` constructor, the `Message.add_payload()` method, the `Utils.dump_address_pair()` function, and the functions `Utils.decode()` and `Utils.encode()`.
- New `DeprecationWarning`s have been added to: `Generator.__call__()`, `Message.get_type()`, `Message.get_main_type()`, `Message.get_subtype()`, and the *strict* argument to the `Parser` class. These are expected to be removed in future versions.
- Support for Pythons earlier than 2.3 has been removed.

Here are the differences between `email` version 2 and version 1:

The `email.Header` and `email.Charset` modules have been added.

The pickle format for `Message` instances has changed. Since this was never (and still isn't) formally defined, this isn't considered a backward incompatibility. However if your application pickles and unpickles `Message` instances, be aware that in `email` version 2, `Message` instances now have private variables *_charset* and *_default_type*.

Several methods in the `Message` class have been deprecated, or their signatures changed. Also, many new methods have been added. See the documentation for the `Message` class for details. The changes should be completely backward compatible.

The object structure has changed in the face of *message/rfc822* content types. In `email` version 1, such a type would be represented by a scalar payload, i.e. the container message's `is_multipart()` returned false, `get_payload()` was not a list object, but a single `Message` instance.

This structure was inconsistent with the rest of the package, so the object representation for *message/rfc822* content types was changed. In `email` version 2, the container *does* return `True` from `is_multipart()`, and `get_payload()` returns a list containing a single `Message` item.

Note that this is one place that backward compatibility could not be completely maintained. However, if you're already testing the return type of `get_payload()`, you should be fine. You just need to make sure your code doesn't do a `set_payload()` with a `Message` instance on a container with a content type of *message/rfc822*.

The `Parser` constructor's *strict* argument was added, and its `parse()` and `parsestr()` methods grew a *headersonly* argument. The *strict* flag was also added to functions `email.message_from_file()` and `email.message_from_string()`.

`Generator.__call__()` is deprecated; use `Generator.flatten()` instead. The `Generator` class has also grown the `clone()` method.

The `DecodedGenerator` class in the `email.Generator` module was added.

The intermediate base classes `MIMENonMultipart` and `MIMEMultipart` have been added, and interposed in the class hierarchy for most of the other MIME-related derived classes.

The *_encoder* argument to the `MIMEText` constructor has been deprecated. Encoding now happens implicitly based on the *_charset* argument.

The following functions in the `email.Utils` module have been deprecated: `dump_address_pairs()`, `decode()`, and `encode()`. The following functions have been added to the module: `make_msgid()`, `decode_rfc2231()`, `encode_rfc2231()`, and `decode_params()`.

The non-public function email.Iterators._structure() was added.

19.1.13. Differences from mimelib¶

The email package was originally prototyped as a separate library called mimelib. Changes have been made so that method names are more consistent, and some methods or modules have either been added or removed. The semantics of some of the methods have also changed. For the most part, any functionality available in mimelib is still available in the email package, albeit often in a different way. Backward compatibility between the mimelib package and the email package was not a priority.

Here is a brief description of the differences between the mimelib and the email packages, along with hints on how to port your applications.

Of course, the most visible difference between the two packages is that the package name has been changed to email. In addition, the top-level package has the following differences:

- messageFromString() has been renamed to message_from_string().
- messageFromFile() has been renamed to message_from_file().

The Message class has the following differences:

- The method asString() was renamed to as_string().
- The method ismultipart() was renamed to is_multipart().
- The get_payload() method has grown a *decode* optional argument.
- The method getall() was renamed to get_all().
- The method addheader() was renamed to add_header().
- The method gettype() was renamed to get_type().
- The method getmaintype() was renamed to get_main_type().
- The method getsubtype() was renamed to get_subtype().
- The method getparams() was renamed to get_params(). Also, whereas getparams() returned a list of strings, get_params() returns a list of 2-tuples, effectively the key/value pairs of the parameters, split on the '=' sign.
- The method getparam() was renamed to get_param().
- The method getcharsets() was renamed to get_charsets().
- The method getfilename() was renamed to get_filename().
- The method getboundary() was renamed to get_boundary().
- The method setboundary() was renamed to set_boundary().
- The method getdecodedpayload() was removed. To get similar functionality, pass the value 1 to the *decode* flag of the get_payload() method.
- The method getpayloadastext() was removed. Similar functionality is supported by the DecodedGenerator class in the email.generator module.
- The method getbodyastext() was removed. You can get similar functionality by creating an iterator with typed_subpart_iterator() in the email.iterators module.

The Parser class has no differences in its public interface. It does have some additional smarts to recognize *message/delivery-status* type messages, which it represents as a Message instance containing separate Message subparts for each header block in the delivery status notification [1].

The Generator class has no differences in its public interface. There is a new class in the email.generator module though, called DecodedGenerator which provides most of the functionality previously available in the Message.getpayloadastext() method.

The following modules and classes have been changed:

The MIMEBase class constructor arguments *_major* and *_minor* have changed to *_maintype* and *_subtype* respectively.

The Image class/module has been renamed to MIMEImage. The *_minor* argument has been renamed to *_subtype*.

The Text class/module has been renamed to MIMEText. The *_minor* argument has been renamed to *_subtype*.

The MessageRFC822 class/module has been renamed to MIMEMessage. Note that an earlier version of mimelib called this class/module RFC822, but that clashed with the Python standard library module rfc822 on some case-insensitive file systems.

Also, the MIMEMessage class now represents any kind of MIME message with main type *message*. It takes an optional argument *_subtype* which is used to set the MIME subtype. *_subtype* defaults to *rfc822*.

mimelib provided some utility functions in its address and date modules. All of these functions have been moved to the email.utils module.

The MsgReader class/module has been removed. Its functionality is most closely supported in the body_line_iterator() function in the email.iterators module.

Footnotes

[1] Delivery Status Notifications (DSN) are defined in **RFC 1894**.

Table Of Contents

Previous topic

19. Internet Data Handling

Next topic

19.1.1. `email`: Representing an email message

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 19. Internet Data Handling »