`!email.contentmanager`: Managing MIME Content

- Content Manager Instances

email.contentmanager

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

3.6[^1]

Base class for content managers. Provides the standard registry mechanisms to register converters between MIME content and other representations, as well as the `get_content` and `set_content` dispatch methods.

get_content(msg, *args,*kw)

Look up a handler function based on the `mimetype` of *msg* (see next paragraph), call it, passing through all arguments, and return the result of the call. The expectation is that the handler will extract the payload from *msg* and return an object that encodes information about the extracted data.

To find the handler, look for the following keys in the registry, stopping with the first one found:

- the string representing the full MIME type (`maintype/subtype`)
- the string representing the `maintype`
- the empty string

If none of these keys produce a handler, raise a `KeyError` for the full MIME type.

set_content(msg, obj, *args,*kw)

If the `maintype` is `multipart`, raise a `TypeError`; otherwise look up a handler function based on the type of *obj* (see next paragraph), call `~email.message.EmailMessage.clear_content` on the *msg*, and call the handler function, passing through all arguments. The expectation is that the handler will transform and store *obj* into *msg*, possibly making other changes to *msg* as well, such as adding various MIME headers to encode information needed to interpret the stored data.

To find the handler, obtain the type of *obj* (`typ = type(obj)`), and look for the following keys in the registry, stopping with the first one found:

- the type itself (`typ`)
- the type's fully qualified name (`typ.__module__ + '.' + typ.__qualname__`).
- the type's `qualname <type.__qualname__>` (`typ.__qualname__`)
- the type's `name <type.__name__>` (`typ.__name__`).

If none of the above match, repeat all of the checks above for each of the types in the `MRO` (`typ.__mro__ <type.__mro__>`). Finally, if no other key yields a handler, check for a handler for the key `None`. If there is no handler for `None`, raise a `KeyError` for the fully qualified name of the type.

Also add a `MIME-Version` header if one is not present (see also `.MIMEPart`).

add_get_handler(key, handler)

Record the function *handler* as the handler for *key*. For the possible values of *key*, see `get_content`.

add_set_handler(typekey, handler)

Record *handler* as the function to call when an object of a type matching *typekey* is passed to `set_content`. For the possible values of *typekey*, see `set_content`.

Content Manager Instances

Currently the email package provides only one concrete content manager, `raw_data_manager`, although more may be added in the future. `raw_data_manager` is the `~email.policy.EmailPolicy.content_manager` provided by `~email.policy.EmailPolicy` and its derivatives.

raw_data_manager

This content manager provides only a minimum interface beyond that provided by `~email.message.Message` itself: it deals only with text, raw bytes, and `~email.message.Message` objects. Nevertheless, it provides significant advantages compared to the base API: `get_content` on a text part will return a string without the application needing to manually decode it, `set_content` provides a rich set of options for controlling the headers added to a part and controlling the content

transfer encoding, and it enables the use of the various `add_` methods, thereby simplifying the creation of multipart messages.

get_content(msg, errors='replace')

Return the payload of the part as either a string (for `text` parts), an `~email.message.EmailMessage` object (for `message/rfc822` parts), or a `bytes` object (for all other non-multipart types). Raise a `KeyError` if called on a `multipart`. If the part is a `text` part and *errors* is specified, use it as the error handler when decoding the payload to a string. The default error handler is `replace`.

set_content(msg, <'str'>, subtype="plain", charset='utf-8', cte=None, disposition=None, filename=None, cid=None, params=None, headers=None) set_content(msg, <'bytes'>, maintype, subtype, cte="base64", disposition=None, filename=None, cid=None, params=None, headers=None) set_content(msg, <'EmailMessage'>, cte=None, disposition=None, filename=None, cid=None, params=None, headers=None)

Add headers and payload to *msg*:

Add a `Content-Type` header with a `maintype/subtype` value.

- For `str`, set the MIME `maintype` to `text`, and set the subtype to *subtype* if it is specified, or `plain` if it is not.
- For `bytes`, use the specified *maintype* and *subtype*, or raise a `TypeError` if they are not specified.
- For `~email.message.EmailMessage` objects, set the maintype to `message`, and set the subtype to *subtype* if it is specified or `rfc822` if it is not. If *subtype* is `partial`, raise an error (`bytes` objects must be used to construct `message/partial` parts).

If *charset* is provided (which is valid only for `str`), encode the string to bytes using the specified character set. The default is `utf-8`. If the specified *charset* is a known alias for a standard MIME charset name, use the standard charset instead.

If *cte* is set, encode the payload using the specified content transfer encoding, and set the `Content-Transfer-Encoding` header to that value. Possible values for *cte* are `quoted-printable`, `base64`, `7bit`, `8bit`, and `binary`. If the input cannot be encoded in the specified encoding (for example, specifying a *cte* of `7bit` for an input that contains non-ASCII values), raise a `ValueError`.

- For `str` objects, if *cte* is not set use heuristics to determine the most compact encoding. Prior to encoding, `str.splitlines` is used to normalize all line boundaries, ensuring that each line of the payload is terminated by the current policy's `~email.policy.Policy.linesep` property (even if the original string did not end with one).
- For `bytes` objects, *cte* is taken to be base64 if not set, and the aforementioned newline translation is not performed.
- For `~email.message.EmailMessage`, per `2046`, raise an error if a *cte* of `quoted-printable` or `base64` is requested for *subtype* `rfc822`, and for any *cte* other than `7bit` for *subtype*

`external-body`. For `message/rfc822`, use `8bit` if *cte* is not specified. For all other values of *subtype*, use `7bit`.

> [!NOTE] A *cte* of `binary` does not actually work correctly yet. The `EmailMessage` object as modified by `set_content` is correct, but `~email.generator.BytesGenerator` does not serialize it correctly.

If *disposition* is set, use it as the value of the `Content-Disposition` header. If not specified, and *filename* is specified, add the header with the value `attachment`. If *disposition* is not specified and *filename* is also not specified, do not add the header. The only valid values for *disposition* are `attachment` and `inline`.

If *filename* is specified, use it as the value of the `filename` parameter of the `Content-Disposition` header.

If *cid* is specified, add a `Content-ID` header with *cid* as its value.

If *params* is specified, iterate its `items` method and use the resulting `(key, value)` pairs to set additional parameters on the `Content-Type` header.

If *headers* is specified and is a list of strings of the form `headername: headervalue` or a list of `header` objects (distinguished from strings by having a `name` attribute), add the headers to *msg*.

Footnotes

[^1]: Originally added in 3.4 as a `provisional module <provisional package>`

□□□ #2
□ □□□ □□□ 2026-10-31 17:08:51 UTC
□ □□□ □□□ 2026-10-31 17:08:54 UTC