`!email.errors`: Exception and Defect classes

email.errors

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

The following exception classes are defined in the `!email.errors` module:

MessageError()

This is the base class for all exceptions that the `email` package can raise. It is derived from the standard `Exception` class and defines no additional methods.

MessageParseError()

This is the base class for exceptions raised by the `~email.parser.Parser` class. It is derived from `MessageError`. This class is also used internally by the parser used by `~email.headerregistry`.

HeaderParseError()

Raised under some error conditions when parsing the `5322` headers of a message, this class is derived from `MessageParseError`. The `~email.message.EmailMessage.set_boundary` method will raise this error if the content type is unknown when the method is called. `~email.header.Header` may raise this error for certain base64 decoding errors, and when an attempt is made to create a header that appears to contain an embedded header (that is, there is what is supposed to be a continuation line that has no leading whitespace and looks like a header).

BoundaryError()

Deprecated and no longer used.

MultipartConversionError()

Raised if the `~email.message.Message.attach` method is called on an instance of a class derived from `~email.mime.nonmultipart.MIMENonMultipart` (e.g. `~email.mime.image.MIMEImage`). `MultipartConversionError` multiply inherits from `MessageError` and the built-in `TypeError`.

HeaderWriteError()

Raised when an error occurs when the `~email.generator` outputs headers.

MessageDefect()

This is the base class for all defects found when parsing email messages. It is derived from `ValueError`.

HeaderDefect()

This is the base class for all defects found when parsing email headers. It is derived from `MessageDefect`.

Here is the list of the defects that the `~email.parser.FeedParser` can find while parsing messages. Note that the defects are added to the message where the problem was found, so for example, if a message nested inside a `multipart/alternative` had a malformed header, that nested message object would have a defect, but the containing messages would not.

All defect classes are subclassed from `email.errors.MessageDefect`.

NoBoundaryInMultipartDefect

A message claimed to be a multipart, but had no `boundary` parameter.

StartBoundaryNotFoundDefect

The start boundary claimed in the `Content-Type` header was never found.

CloseBoundaryNotFoundDefect

A start boundary was found, but no corresponding close boundary was ever found.

3.3

FirstHeaderLineIsContinuationDefect

The message had a continuation line as its first header line.

MisplacedEnvelopeHeaderDefect

A "Unix From" header was found in the middle of a header block.

MissingHeaderBodySeparatorDefect

A line was found while parsing headers that had no leading white space but contained no ':'. Parsing continues assuming that the line represents the first line of the body.

3.3

MalformedHeaderDefect

A header was found that was missing a colon, or was otherwise malformed.

3.3 This defect has not been used for several Python versions.

MultipartInvariantViolationDefect

A message claimed to be a `multipart`, but no subparts were found. Note that when a message has this defect, its `~email.message.Message.is_multipart` method may return `False` even though its content type claims to be `multipart`.

InvalidBase64PaddingDefect

When decoding a block of base64 encoded bytes, the padding was not correct. Enough padding is added to perform the decode, but the resulting decoded bytes may be invalid.

InvalidBase64CharactersDefect

When decoding a block of base64 encoded bytes, characters outside the base64 alphabet were encountered. The characters are ignored, but the resulting decoded bytes may be invalid.

InvalidBase64LengthDefect

When decoding a block of base64 encoded bytes, the number of non-padding base64 characters was invalid (1 more than a multiple of 4). The encoded block was kept as-is.

InvalidDateDefect

When decoding an invalid or unparsable date field. The original value is kept as-is.

□□□ #2
□ □□□ □□□ 2026-10-31 17:08:58 UTC
□ □□□ □□□ 2026-10-31 17:09:00 UTC