19.10. `multifile` — Support for files containing distinct parts¶

Deprecated since version 2.5: The `email` package should be used in preference to the `multifile` module. This module is present only to maintain backward compatibility.

The `MultiFile` object enables you to treat sections of a text file as file-like input objects, with `''` being returned by `readline()` when a given delimiter pattern is encountered. The defaults of this class are designed to make it useful for parsing MIME multipart messages, but by subclassing it and overriding methods it can be easily adapted for more general use.

class `multifile.MultiFile`(*fp*[, *seekable*])¶

Create a multi-file. You must instantiate this class with an input object argument for the `MultiFile` instance to get lines from, such as a file object returned by `open()`.

`MultiFile` only ever looks at the input object's `readline()`, `seek()` and `tell()` methods, and the latter two are only needed if you want random access to the individual MIME parts. To use `MultiFile` on a non-seekable stream object, set the optional *seekable* argument to false; this will prevent using the input object's `seek()` and `tell()` methods.

It will be useful to know that in `MultiFile`'s view of the world, text is composed of three kinds of lines: data, section-dividers, and end-markers. MultiFile is designed to support parsing of messages that may have multiple nested message parts, each with its own pattern for section-divider and end-marker lines.

See also

Module `email`
Comprehensive email handling package; supersedes the `multifile` module.

19.10.1. MultiFile Objects¶

A `MultiFile` instance has the following methods:

`MultiFile.readline`(*str*)¶
Read a line. If the line is data (not a section-divider or end-marker or real EOF) return it. If the line matches the most-recently-stacked boundary, return `''` and set `self.last` to 1 or 0 according as the match is or is not an end-marker. If the line matches any other stacked boundary, raise an error. On encountering end-of-file on the underlying stream object, the method raises `Error` unless all boundaries have been popped.

`MultiFile.readlines`(*str*)¶
Return all lines remaining in this part as a list of strings.

`MultiFile.read`()¶
Read all lines, up to the next section. Return them as a single (multiline) string. Note that this doesn't take a size argument!

`MultiFile.seek`(*pos*[, *whence*])¶
Seek. Seek indices are relative to the start of the current section. The *pos* and *whence* arguments are interpreted as for a file seek.

`MultiFile.tell`()¶
Return the file position relative to the start of the current section.

`MultiFile.next`()¶
Skip lines to the next section (that is, read lines until a section-divider or end-marker has been consumed). Return true if there is such a section, false if an end-marker is seen. Re-enable the most-recently-pushed boundary.

`MultiFile.is_data`(*str*)¶

Return true if *str* is data and false if it might be a section boundary. As written, it tests for a prefix other than `'--'` at start of line (which all MIME boundaries have) but it is declared so it can be overridden in derived classes.

Note that this test is used intended as a fast guard for the real boundary tests; if it always returns false it will merely slow processing, not cause it to fail.

`MultiFile.push`(*str*)¶

Push a boundary string. When a decorated version of this boundary is found as an input line, it will be interpreted as a section-divider or end-marker (depending on the decoration, see **RFC 2045**). All subsequent reads will return the empty string to indicate end-of-file, until a call to `pop()` removes the boundary a or `next()` call reenables it.

It is possible to push more than one boundary. Encountering the most-recently-pushed boundary will return EOF; encountering any other boundary will raise an error.

`MultiFile.pop`()¶
Pop a section boundary. This boundary will no longer be interpreted as EOF.

`MultiFile.section_divider`(*str*)¶
Turn a boundary into a section-divider line. By default, this method prepends `'--'` (which MIME section boundaries have) but it is declared so it can be overridden in derived classes. This method need not append LF or CR-LF, as comparison with the result ignores trailing whitespace.

`MultiFile.end_marker`(*str*)¶
Turn a boundary string into an end-marker line. By default, this method prepends `'--'` and appends `'--'` (like a MIME-multipart end-of-message marker) but it is declared so it can be overridden in derived classes. This method need not append LF or CR-LF, as comparison with the result ignores trailing whitespace.

Finally, `MultiFile` instances have two public instance variables:

`MultiFile.level`¶
Nesting depth of the current part.

`MultiFile.last`¶
True if the last end-of-file was for an end-of-message marker.

19.10.2. `MultiFile` Example¶

```
import mimetools
import multifile
import StringIO

def extract_mime_part_matching(stream, mimetype):
   """Return the first element in a multipart MIME message on stream
   matching mimetype."""

   msg = mimetools.Message(stream)
   msgtype = msg.gettype()
   params = msg.getplist()

   data = StringIO.StringIO()
   if msgtype[:10] == "multipart/":

       file = multifile.MultiFile(stream)
       file.push(msg.getparam("boundary"))
       while file.next():
           submsg = mimetools.Message(file)
           try:
               data = StringIO.StringIO()
               mimetools.decode(file, data, submsg.getencoding())
           except ValueError:
               continue
           if submsg.gettype() == mimetype:
               break
       file.pop()
   return data.getvalue()
```


This Page

- Show Source