Mail and News Message Processing

"To be removed from our list of future commercial postings by [SOME] PUBLISHING COMPANY an Annual Charge of Ninety Five dollars is required. Just send $95.00 with your Name, Address and Name of the Newsgroup to be removed from our list"

newsgroup spammer, July 1996

Overview

Python comes with a rich set of modules for processing mail and news messages, as well as some common mail archive (mailbox) formats.

The rfc822 module

This module contains a parser for mail and news messages (and any other message that conforms to the RFC 822 standard, such as HTTP headers).

Basically, an RFC 822 style message consists of a number of header fields, followed by at least one blank line, and the message body itself.

For example, here's a short mail message. The first five lines make up the message header, and the actual messages (a single line, in this case) follows after an empty line:

```
Message-Id: <20001114144603.00abb310@oreilly.com>
Date: Tue, 14 Nov 2000 14:55:07 -0500
To: "Fredrik Lundh" <fredrik@effbot.org>
From: Frank
Subject: Re: python library book!

Where is it?
```

The message parser reads the headers, and returns a dictionary-like object, with the message headers as keys.

Example: Using the rfc822 module

```
# File:rfc822-example-1.py

import rfc822

file = open("samples/sample.eml")

message = rfc822.Message(file)

for k, v in message.items():
    print k, "=", v

print len(file.read()), "bytes in body"
```

```
subject = Re: python library book!
from = "Frank" <your@editor>
message-id = <20001114144603.00abb310@oreilly.com>
to = "Fredrik Lundh" <fredrik@effbot.org>
date = Tue, 14 Nov 2000 14:55:07 -0500
25 bytes in body
```

The message object also provides a couple of convenience methods, which parses address fields and dates for you:

Example: Parsing header fields using the rfc822 module

```
# File:rfc822-example-2.py

import rfc822

file = open("samples/sample.eml")

message = rfc822.Message(file)

print message.getdate("date")
print message.getaddr("from")
print message.getaddrlist("to")
```

```
(2000, 11, 14, 14, 55, 7, 0, 0, 0)
('Frank', 'your@editor')
[('Fredrik Lundh', 'fredrik@effbot.org')]
```

The address fields are parsed into (mail, real name)-tuples. The date field is parsed into a 9-element time tuple, ready for use with the **time** module.

The mimetools module

The *Multipurpose Internet Mail Extensions* (MIME) standard defines how to store non-ASCII text, images, and other data in RFC 822-style messages.

This module contains a number of tools for writing programs which read or write MIME messages. Among other things, it contains a version of the **rfc822** module's **Message** class, which knows a bit more about MIME encoded messages.

Example: Using the mimetools module

```
# File:mimetools-example-1.py

import mimetools

file = open("samples/sample.msg")

msg = mimetools.Message(file)

print "type", "=>", msg.gettype()
print "encoding", "=>", msg.getencoding()
print "plist", "=>", msg.getplist()

print "header", "=>"
for k, v in msg.items():
    print "  ", k, "=", v
```

```
type => text/plain
encoding => 7bit
plist => ['charset="iso-8859-1"']
header =>
  mime-version = 1.0
  content-type = text/plain;
 charset="iso-8859-1"
  to = effbot@spam.egg
  date = Fri, 15 Oct 1999 03:21:15 -0400
  content-transfer-encoding = 7bit
  from = "Fredrik Lundh" <fredrik@pythonware.com>
  subject = By the way...
...
```

The MimeWriter module

This module can be used to write "multipart" messages, as defined by the MIME mail standard.

Example: Using the MimeWriter module

```
# File:mimewriter-example-1.py

import MimeWriter

# data encoders
import quopri
import base64
import StringIO

import sys

TEXT = """
here comes the image you asked for.  hope
it's what you expected.

</F>"""

FILE = "samples/sample.jpg"

file = sys.stdout

#
# create a mime multipart writer instance

mime = MimeWriter.MimeWriter(file)
mime.addheader("Mime-Version", "1.0")

mime.startmultipartbody("mixed")

# add a text message

part = mime.nextpart()
part.addheader("Content-Transfer-Encoding", "quoted-printable")
part.startbody("text/plain")

quopri.encode(StringIO.StringIO(TEXT), file, 0)

# add an image

part = mime.nextpart()
part.addheader("Content-Transfer-Encoding", "base64")
part.startbody("image/jpeg")

base64.encode(open(FILE, "rb"), file)

mime.lastpart()
```

The output looks something like:

```
Content-Type: multipart/mixed;
   boundary='host.1.-852461.936831373.130.24813'

--host.1.-852461.936831373.130.24813
Content-Type: text/plain
Context-Transfer-Encoding: quoted-printable

here comes the image you asked for.  hope
it's what you expected.

</F>

--host.1.-852461.936831373.130.24813
Content-Type: image/jpeg
Context-Transfer-Encoding: base64

/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8
UHRofHh0a
HBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2wBDAQkJCQwLDBgNDRgyIRwhM
jIyMjIy
...
1e5vLrSYbJnEVpEgjCLx5mPU0qsVK0UaxjdNlS+1U6pfzTR8IzEhj2HrVG6m8m18xc8cIKSC
Aysl
tCuFyC746j/Cq2pTia4WztfmKjGBXTCmo6IUptn/2Q==

--host.1.-852461.936831373.130.24813--
```

Here's a larger example, which uses a helper class that stores each subpart in the most suitable way:

Example: A helper class for the MimeWriter module

```
# File:mimewriter-example-2.py

import MimeWriter
import string, StringIO, sys
import re, quopri, base64

# check if string contains non-ascii characters
must_quote = re.compile("[\177-\377]").search


#
# encoders

def encode_quoted_printable(infile, outfile):
   quopri.encode(infile, outfile, 0)
```

```
class Writer:

    def __init__(self, file=None, blurb=None):
        if file is None:
            file = sys.stdout
        self.file = file
        self.mime = MimeWriter.MimeWriter(file)
        self.mime.addheader("Mime-Version", "1.0")

        file = self.mime.startmultipartbody("mixed")
        if blurb:
            file.write(blurb)

    def close(self):
        "End of message"
        self.mime.lastpart()
        self.mime = self.file = None

    def write(self, data, mimetype="text/plain"):
        "Write data from string or file to message"

        # data is either an opened file or a string
        if type(data) is type(""):
            file = StringIO.StringIO(data)
        else:
            file = data
            data = None

        part = self.mime.nextpart()

        typ, subtyp = string.split(mimetype, "/", 1)

        if typ == "text":

            # text data
            encoding = "quoted-printable"
            encoder = lambda i, o: quopri.encode(i, o, 0)

            if data and not must_quote(data):
                # copy, don't encode
                encoding = "7bit"
                encoder = None

        else:

            # binary data (image, audio, application, ...)
            encoding = "base64"
            encoder = base64.encode
```

```
        #
        # write part headers

        if encoding:
            part.addheader("Content-Transfer-Encoding", encoding)

        part.startbody(mimetype)

        #
        # write part body

        if encoder:
            encoder(file, self.file)
        elif data:
            self.file.write(data)
        else:
            while 1:
                data = infile.read(16384)
                if not data:
                    break
                outfile.write(data)

#
# try it out

BLURB = "if you can read this, your mailer is not MIME-aware\n"

mime = Writer(sys.stdout, BLURB)

# add a text message
mime.write("""\
here comes the image you asked for.  hope
it's what you expected.
""", "text/plain")

# add an image
mime.write(open("samples/sample.jpg", "rb"), "image/jpeg")

mime.close()
```

The mailbox module

This module contains code to deal with a number of different mailbox formats (mostly Unix formats). Most mailbox formats simply store plain RFC 822-style messages in a long text file, using some kind of separator line to tell one message from another.

Example: Using the mailbox module

```
# File:mailbox-example-1.py

import mailbox

mb = mailbox.UnixMailbox(open("/var/spool/mail/effbot"))

while 1:
    msg = mb.next()
    if not msg:
        break
    for k, v in msg.items():
        print k, "=", v
    body = msg.fp.read()
    print len(body), "bytes in body"
```

```
subject = for he's a ...
message-id = <199910150027.CAA03202@spam.egg>
received = (from fredrik@pythonware.com)
 by spam.egg (8.8.7/8.8.5) id CAA03202
 for effbot; Fri, 15 Oct 1999 02:27:36 +0200
from = Fredrik Lundh <fredrik@pythonware.com>
date = Fri, 15 Oct 1999 12:35:36 +0200
to = effbot@spam.egg
1295 bytes in body
```

The mailcap module

This module contains code to deal with "mailcap" files, which contain information on how to deal with different document formats (on Unix platforms).

Example: Using the mailcap module to get a capability dictionary

```
# File:mailcap-example-1.py

import mailcap

caps = mailcap.getcaps()

for k, v in caps.items():
    print k, "=", v
```

```
image/* = [{'view': 'pilview'}]
application/postscript = [{'view': 'ghostview'}]
```

In the above example, the system uses **pilview** for all kinds of images, and **ghostscript** viewer for PostScript documents.

Example: Using the mailcap module to find a viewer

```
# File:mailcap-example-2.py

import mailcap

caps = mailcap.getcaps()

command, info = mailcap.findmatch(
    caps, "image/jpeg", "view", "samples/sample.jpg"
    )

print command
```

```
pilview samples/sample.jpg
```

The mimetypes module

This module contains support for determining the MIME type for a given uniform resource locator. This is based on a built-in table, plus Apache and Netscape configuration files, if they are found.

Example: Using the mimetypes module

```
# File:mimetypes-example-1.py

import mimetypes
import glob, urllib

for file in glob.glob("samples/*"):
    url = urllib.pathname2url(file)
    print file, mimetypes.guess_type(url)
```

```
samples\sample.au ('audio/basic', None)
samples\sample.ini (None, None)
samples\sample.jpg ('image/jpeg', None)
samples\sample.msg (None, None)
samples\sample.tar ('application/x-tar', None)
samples\sample.tgz ('application/x-tar', 'gzip')
samples\sample.txt ('text/plain', None)
samples\sample.wav ('audio/x-wav', None)
samples\sample.zip ('application/zip', None)
```

The packmail module

(Obsolete) This module contain tools to create Unix "shell archives". If you have the right tools installed (if you have a Unix box, they are installed), you can unpack such an archive simply by executing it.

Example: Using the packmail module to pack a single file

```
# File:packmail-example-1.py

import packmail
import sys

packmail.pack(sys.stdout, "samples/sample.txt", "sample.txt")
```

```
echo sample.txt
sed "s/^X//" >sample.txt <<"!"
XWe will perhaps eventually be writing only small
Xmodules which are identified by name as they are
Xused to build larger ones, so that devices like
Xindentation, rather than delimiters, might become
Xfeasible for expressing local structure in the
Xsource language.
X    -- Donald E. Knuth, December 1974
!
```

Example: Using the packmail module to pack an entire directory tree

```
# File:packmail-example-2.py

import packmail
import sys

packmail.packtree(sys.stdout, "samples")
```

Note that this module cannot handle binary files, such as images and sound snippets.

The mimify module

This module converts MIME encoded text messages from encoded formats to plain text (typically ISO Latin), and back. It can be used as a command line tool, and as a conversion filter for certain mail agents.

```
$ mimify.py -e raw-message mime-message
$ mimify.py -d mime-message raw-message
```

It can also be used as a module, as shown in the following example:

Example: Using the mimify module to decode a message

```
# File:mimify-example-1.py

import mimify
import sys

mimify.unmimify("samples/sample.msg", sys.stdout, 1)
```

Here's a MIME message containing two parts, one encoded as quoted-printable, and the other as base64. The third argument to **unmimify** controls whether base64-encoded parts should be decoded or not.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary='boundary'

this is a multipart sample file.  the two
parts both contain ISO Latin 1 text, with
different encoding techniques.

--boundary
Content-Type: text/plain
Content-Transfer-Encoding: quoted-printable

sillmj=F6lke! blindstyre! medisterkorv!

--boundary
Content-Type: text/plain
Content-Transfer-Encoding: base64

a29tIG5lciBiYXJhLCBvbSBkdSB09nJzIQ==

--boundary--
```

And here's the decoded result. Much more readable, at least if you know the language.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary='boundary'

this is a multipart sample file.  the two
parts both contain ISO Latin 1 text, with
different encoding techniques.

--boundary
Content-Type: text/plain

sillmjölke! blindstyre! medisterkorv!

--boundary
Content-Type: text/plain

kom ner bara, om du törs!
```

Encoding messages is as easy:

Example: Using the mimify module to encode a message

```
# File:mimify-example-2.py

import mimify
import StringIO, sys

#
# decode message into a string buffer

file = StringIO.StringIO()

mimify.unmimify("samples/sample.msg", file, 1)

#
# encode message from string buffer

file.seek(0) # rewind

mimify.mimify(file, sys.stdout)
```

The multifile module

A support module that allows you to treat each part of a multipart MIME message as an individual file.

Example: Using the multifile module

```
# File:multifile-example-1.py

import multifile
import cgi, rfc822

infile = open("samples/sample.msg")

message = rfc822.Message(infile)

# print parsed header
for k, v in message.items():
    print k, "=", v

# use cgi support function to parse content-type header
type, params = cgi.parse_header(message["content-type"])

if type[:10] == "multipart/":

    # multipart message
    boundary = params["boundary"]

    file = multifile.MultiFile(infile)

    file.push(boundary)

    while file.next():

        submessage = rfc822.Message(file)

        # print submessage
        print "-" * 68
        for k, v in submessage.items():
            print k, "=", v
        print
        print file.read()

    file.pop()

else:

    # plain message
    print infile.read()
```