had to processes a bunch of emails for a $DAYJOB task this week and my “default setting” is to use R for pretty much everything (this should come as no surprise). Treating mail as data is not an uncommon task and many R packages exist that can reach out and grab mail from servers or work directly with local mail archives.

**Mbox’in off the rails on a crazy tm**[**1**](https://rud.is/b/2018/08/04/digging-into-mbox-details-a-tale-of-tm-reticulate/#f1)

This particular mail corpus is in [mbox[](https://www.loc.gov/preservation/digital/formats/fdd/fdd000383.shtml)](https://www.loc.gov/preservation/digital/formats/fdd/fdd000383.shtml) format since it was saved via Apple Mail. It’s one big text file with each message appearing one after the other. The format has been around for decades, and R’s tm package — via the tm.plugin.mail plugin package — can process these mbox files.

To demonstrate, we’ll use an Apple Mail archive excerpt from a set of R mailing list messages as they are not private/sensitive:

library(tm)

library(tm.plugin.mail)

# point the tm corpus machinery to the mbox file and let it know the timestamp format since it varies

VCorpus(

MBoxSource("~/Data/test.mbox/mbox"),

readerControl = list(

reader = readMail(DateFormat = "%a, %e %b %Y %H:%M:%S %z")

)

) -> mbox

str(unclass(mbox), 1)

## List of 3

## $ content:List of 198

## $ meta : list()

## ..- attr(\*, "class")= chr "CorpusMeta"

## $ dmeta :'data.frame': 198 obs. of 0 variables

str(unclass(mbox[[1]]), 1)

## List of 2

## $ content: chr [1:476] "Try this:" "" "> library(lubridate)" "> library(tidyverse)" ...

## $ meta :List of 9

## ..- attr(\*, "class")= chr "TextDocumentMeta"

str(unclass(mbox[[1]]$meta), 1)

## List of 9

## $ author : chr "jim holtman "

## $ datetimestamp: POSIXlt[1:1], format: "2018-08-01 15:01:17"

## $ description : chr(0)

## $ heading : chr "Re: [R] read txt file - date - no space"

## $ id : chr ""

## $ language : chr "en"

## $ origin : chr(0)

## $ header : chr [1:145] "Delivered-To: bob@rud.is" "Received: by 2002:ac0:e681:0:0:0:0:0 with SMTP id b1-v6csp950182imq;" " Wed, 1 Aug 2018 08:02:23 -0700 (PDT)" "X-Google-Smtp-Source: AAOMgpcdgBD4sDApBiF2DpKRfFZ9zi/4Ao32Igz9n8vT7EgE6InRoa7VZelMIik7OVmrFCRPDBde" ...

## $ : NULL

We’re using unclass() since the str() output gets a bit crowded with all of the tm class attributes stuck in the output display.

The tm suite is designed for text mining. My task had nothing to do with text mining and I really just needed some header fields and body content in a data frame. If you’ve been working with R for a while, some things in the str() output will no doubt cause a bit of angst. For instance:

* datetimestamp: POSIXlt[1:1], : POSIXlt and data frames really don’t mix well
* description : chr(0) / origin : chr(0): zero-length character vectors 
* $ : NULL : Blank element name with a NULL value…*I Don’t Even* [2](https://rud.is/b/2018/08/04/digging-into-mbox-details-a-tale-of-tm-reticulate/#f2)

The tm suite is also *super opinionated* and “helpfully” left out a ton of headers (though it did keep the source for the complete headers around). Still, we can roll up our sleeves and turn that into a data frame:

# helper function for cleaner/shorter code

`%|0|%` <- function(x, y) { if (length(x) == 0) y else x }

# might as well stay old-school since we're using tm

do.call(

rbind.data.frame,

lapply(mbox, function(.x) {

# we have a few choices, but this one is pretty explicit abt what it does

# so we'll likely be able to decipher it quickly in 2 years when/if we come

# back to it

data.frame(

author = .x$meta$author %|0|% NA\_character\_,

datetimestamp = as.POSIXct(.x$meta$datetimestamp %|0|% NA),

description = .x$meta$description %|0|% NA\_character\_,

heading = .x$meta$heading %|0|% NA\_character\_,

id = .x$meta$id %|0|% NA\_character\_,

language = .x$meta$language %|0|% NA\_character\_,

origin = .x$meta$origin %|0|% NA\_character\_,

header = I(list(.x$meta$header %|0|% NA\_character\_)),

body = I(list(.x$content %|0|% NA\_character\_)),

stringsAsFactors = FALSE

)

})

) %>%

glimpse()

## Observations: 198

## Variables: 9

## $ author "jim holtman ", "PIKAL Petr ...

## $ datetimestamp 2018-08-01 15:01:17, 2018-08-01 13:09:18, 2018-...

## $ description NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...

## $ heading "Re: [R] read txt file - date - no space", "Re: ...

## $ id " "en", "en", "en", "en", "en", "en", "en", "en", ...

## $ origin NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...

## $ header Delivere...., Delivere...., Delivere...., De...

## $ body Try this...., SGkNCg0K...., Dear Pik...., De...

That wasn’t a huge effort, but we would now have to re-process the headers and/or write a custom version of tm.plugin.mail::readMail() (the function source is very readable and extendable) to get any extra data out. Here’s what that might look like:

# Custom msg reader

read\_mail <- function(elem, language, id) {

# extract header val

hdr\_val <- function(src, pat) {

gsub(

sprintf("%s: ", pat), "",

grep(sprintf("^%s:", pat), src, "", value = TRUE, useBytes = TRUE)

) %|0|% NA

}

mail <- elem$content

index <- which(mail == "")[1]

header <- mail[1:index]

mid <- hdr\_val(header, "Message-ID")

PlainTextDocument(

x = mail[(index + 1):length(mail)],

author = hdr\_val(header, "From"),

spam\_score = hdr\_val(header, "X-Spam-Score"), ### <<==== an extra header!

datetimestamp = as.POSIXct(hdr\_val(header, "Date"), format = "%a, %e %b %Y %H:%M:%S %z", tz = "GMT"),

description = NA\_character\_,

header = header,

heading = hdr\_val(header, "Subject"),

id = if (length(mid)) mid[1] else id,

language = language,

origin = hdr\_val(header, "Newsgroups"),

class = "MailDocument"

)

}

VCorpus(

MBoxSource("~/Data/test.mbox/mbox"),

readerControl = list(reader = read\_mail)

) -> mbox

str(unclass(mbox[[1]]$meta), 1)

## List of 9

## $ author : chr "jim holtman "

## $ datetimestamp: POSIXct[1:1], format: "2018-08-01 15:01:17"

## $ description : chr NA

## $ heading : chr "Re: [R] read txt file - date - no space"

## $ id : chr ""

## $ language : chr "en"

## $ origin : chr NA

## $ spam\_score : chr "-3.631"

## $ header : chr [1:145] "Delivered-To: bob@rud.is" "Received: by 2002:ac0:e681:0:0:0:0:0 with SMTP id b1-v6csp950182imq;" " Wed, 1 Aug 2018 08:02:23 -0700 (PDT)" "X-Google-Smtp-Source: AAOMgpcdgBD4sDApBiF2DpKRfFZ9zi/4Ao32Igz9n8vT7EgE6InRoa7VZelMIik7OVmrFCRPDBde" ...

If we wanted all the headers, there are even more succinct ways to solve for that use case.

**Packaging up emails with a reticulated message.mbox**

Since the default functionality of tm.plugin.mail::readMail() forced us to work a bit to get what we needed there’s some justification in seeking out an alternative path. I’ve [written about reticulate before](https://rud.is/b/?s=reticulate) and am including it in this post as the Python standard library module [mailbox[](https://docs.python.org/3.7/library/mailbox.html?highlight=mailbox#mbox)](https://docs.python.org/3.7/library/mailbox.html?highlight=mailbox#mbox) can also make quick work of mbox files.

Two pieces of advice I generally reiterate when I talk about reticulate is that I highly recommend using Python 3 (remember, it’s a fragmented ecosystem) and that I prefer specifying the specific target Python to use via the RETICULATE\_PYTHON environment variable that I have in ~/.Renviron as RETICULATE\_PYTHON=/usr/local/bin/python3.

Let’s bring the mailbox module into R:

library(reticulate)

library(tidyverse)

mailbox <- import("mailbox")

If you're unfamiliar with a Python module or object, you can get help right in R via reticulate::py\_help(). Et sequitur[3](https://rud.is/b/2018/08/04/digging-into-mbox-details-a-tale-of-tm-reticulate/#f3): py\_help(mailbox) will bring up the text help for that module and py\_help(mailbox$mbox) (remember, we swap out dots for dollars when referencing Python object components in R) will do the same for the mailbox.mbox class.

Text help is great and all, but we can also render it to HTML with this helper function:

py\_doc <- function(x) {

require("htmltools")

require("reticulate")

pydoc <- reticulate::import("pydoc")

htmltools::html\_print(

htmltools::HTML(

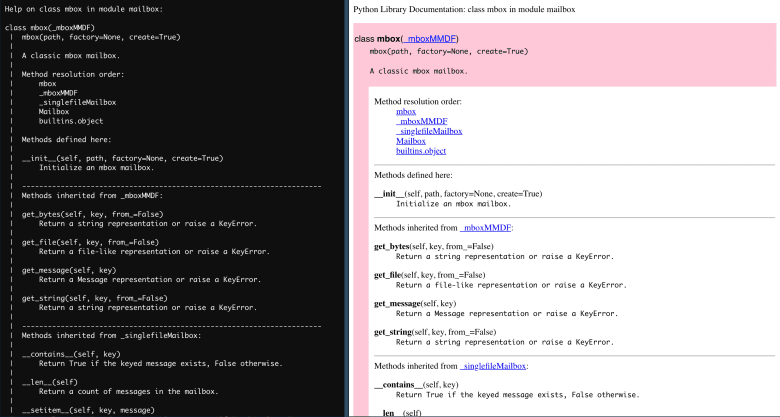
pydoc$render\_doc(x, renderer=pydoc$HTMLDoc())

)

)

}

Here's what the text and HTML help for mailbox.mbox look side-by-side:

[](https://rud.is/b/2018/08/04/digging-into-mbox-details-a-tale-of-tm-reticulate/py-help-doc/)

We can also use a helper function to view the [online documentation](https://docs.python.org/):

readthedocs <- function(obj, py\_ver=3, check\_keywords = "yes") {

require("glue")

query <- obj$`\_\_name\_\_`

browseURL(

glue::glue(

"https://docs.python.org/{py\_ver}/search.html?q={query}&check\_keywords={check\_keywords}"

)

)

}

Et sequitur: readthedocs(mailbox$mbox) will take us to [this results page](https://docs.python.org/3/search.html?q=mbox&check_keywords=yes)

Going back to the task at hand, we need to cycle through the messages and make a data frame for the bits we (well, *I*, care about). The reticulate package does an amazing job making Python objects first-class citizens in R, but Python objects may feel "opaque" to R users since we have to use the $ syntax to get to methods and values and — very often — familiar helpers such as str() are less than helpful on these objects. Let's try to look at the first message (remember, Python is 0-indexed):

msg1 <- mbox$get(0)

str(msg1)

msg1

The output for those last two calls is not shown because they both are just a *large* text dump of the message source. #unhelpful

We *can* get more details, and we'll wrap some punctuation-filled calls in two, small helper functions that have names that will sound familiar:

pstr <- function(obj, ...) { str(obj$`\_\_dict\_\_`, ...) } # like 'str()`

pnames <- function(obj) { import\_builtins()$dir(obj) } # like 'names()' but more complete

Lets see them in action:

pstr(msg1, 1) # we can pass any params str() will take

## List of 10

## $ \_from : chr "jholtman@gmail.com Wed Aug 01 15:02:23 2018"

## $ policy :Compat32()

## $ \_headers :List of 56

## $ \_unixfrom : NULL

## $ \_payload : chr "Try this:\n\n> library(lubridate)\n> library(tidyverse)\n> input <- read.csv(text =3D \"date,str1,str2,str3\n+ "| \_\_truncated\_\_

## $ \_charset : NULL

## $ preamble : NULL

## $ epilogue : NULL

## $ defects : list()

## $ \_default\_type: chr "text/plain"

pnames(msg1)

## [1] "\_\_bytes\_\_" "\_\_class\_\_"

## [3] "\_\_contains\_\_" "\_\_delattr\_\_"

## [5] "\_\_delitem\_\_" "\_\_dict\_\_"

## [7] "\_\_dir\_\_" "\_\_doc\_\_"

## [9] "\_\_eq\_\_" "\_\_format\_\_"

## [11] "\_\_ge\_\_" "\_\_getattribute\_\_"

## [13] "\_\_getitem\_\_" "\_\_gt\_\_"

## [15] "\_\_hash\_\_" "\_\_init\_\_"

## [17] "\_\_init\_subclass\_\_" "\_\_iter\_\_"

## [19] "\_\_le\_\_" "\_\_len\_\_"

## [21] "\_\_lt\_\_" "\_\_module\_\_"

## [23] "\_\_ne\_\_" "\_\_new\_\_"

## [25] "\_\_reduce\_\_" "\_\_reduce\_ex\_\_"

## [27] "\_\_repr\_\_" "\_\_setattr\_\_"

## [29] "\_\_setitem\_\_" "\_\_sizeof\_\_"

## [31] "\_\_str\_\_" "\_\_subclasshook\_\_"

## [33] "\_\_weakref\_\_" "\_become\_message"

## [35] "\_charset" "\_default\_type"

## [37] "\_explain\_to" "\_from"

## [39] "\_get\_params\_preserve" "\_headers"

## [41] "\_payload" "\_type\_specific\_attributes"

## [43] "\_unixfrom" "add\_flag"

## [45] "add\_header" "as\_bytes"

## [47] "as\_string" "attach"

## [49] "defects" "del\_param"

## [51] "epilogue" "get"

## [53] "get\_all" "get\_boundary"

## [55] "get\_charset" "get\_charsets"

## [57] "get\_content\_charset" "get\_content\_disposition"

## [59] "get\_content\_maintype" "get\_content\_subtype"

## [61] "get\_content\_type" "get\_default\_type"

## [63] "get\_filename" "get\_flags"

## [65] "get\_from" "get\_param"

## [67] "get\_params" "get\_payload"

## [69] "get\_unixfrom" "is\_multipart"

## [71] "items" "keys"

## [73] "policy" "preamble"

## [75] "raw\_items" "remove\_flag"

## [77] "replace\_header" "set\_boundary"

## [79] "set\_charset" "set\_default\_type"

## [81] "set\_flags" "set\_from"

## [83] "set\_param" "set\_payload"

## [85] "set\_raw" "set\_type"

## [87] "set\_unixfrom" "values"

## [89] "walk"

names(msg1)

## [1] "add\_flag" "add\_header"

## [3] "as\_bytes" "as\_string"

## [5] "attach" "defects"

## [7] "del\_param" "epilogue"

## [9] "get" "get\_all"

## [11] "get\_boundary" "get\_charset"

## [13] "get\_charsets" "get\_content\_charset"

## [15] "get\_content\_disposition" "get\_content\_maintype"

## [17] "get\_content\_subtype" "get\_content\_type"

## [19] "get\_default\_type" "get\_filename"

## [21] "get\_flags" "get\_from"

## [23] "get\_param" "get\_params"

## [25] "get\_payload" "get\_unixfrom"

## [27] "is\_multipart" "items"

## [29] "keys" "policy"

## [31] "preamble" "raw\_items"

## [33] "remove\_flag" "replace\_header"

## [35] "set\_boundary" "set\_charset"

## [37] "set\_default\_type" "set\_flags"

## [39] "set\_from" "set\_param"

## [41] "set\_payload" "set\_raw"

## [43] "set\_type" "set\_unixfrom"

## [45] "values" "walk"

# See the difference between pnames() and names()

setdiff(pnames(msg1), names(msg1))

## [1] "\_\_bytes\_\_" "\_\_class\_\_"

## [3] "\_\_contains\_\_" "\_\_delattr\_\_"

## [5] "\_\_delitem\_\_" "\_\_dict\_\_"

## [7] "\_\_dir\_\_" "\_\_doc\_\_"

## [9] "\_\_eq\_\_" "\_\_format\_\_"

## [11] "\_\_ge\_\_" "\_\_getattribute\_\_"

## [13] "\_\_getitem\_\_" "\_\_gt\_\_"

## [15] "\_\_hash\_\_" "\_\_init\_\_"

## [17] "\_\_init\_subclass\_\_" "\_\_iter\_\_"

## [19] "\_\_le\_\_" "\_\_len\_\_"

## [21] "\_\_lt\_\_" "\_\_module\_\_"

## [23] "\_\_ne\_\_" "\_\_new\_\_"

## [25] "\_\_reduce\_\_" "\_\_reduce\_ex\_\_"

## [27] "\_\_repr\_\_" "\_\_setattr\_\_"

## [29] "\_\_setitem\_\_" "\_\_sizeof\_\_"

## [31] "\_\_str\_\_" "\_\_subclasshook\_\_"

## [33] "\_\_weakref\_\_" "\_become\_message"

## [35] "\_charset" "\_default\_type"

## [37] "\_explain\_to" "\_from"

## [39] "\_get\_params\_preserve" "\_headers"

## [41] "\_payload" "\_type\_specific\_attributes"

## [43] "\_unixfrom"

Using just names() excludes the "hidden" builtins for Python objects, but knowing they are there and what they are can be helpful, depending on the program context.

Let's continue on the path to our messaging goal and see what headers are available. We'll use some domain knowledge about the \_headers component, though we won't end up going that route to build a data frame:

map\_chr(msg1$`\_headers`, ~.x[[1]])

## [1] "Delivered-To" "Received"

## [3] "X-Google-Smtp-Source" "X-Received"

## [5] "ARC-Seal" "ARC-Message-Signature"

## [7] "ARC-Authentication-Results" "Return-Path"

## [9] "Received" "Received-SPF"

## [11] "Authentication-Results" "Received"

## [13] "X-Virus-Scanned" "Received"

## [15] "Received" "Received"

## [17] "X-Virus-Scanned" "X-Spam-Flag"

## [19] "X-Spam-Score" "X-Spam-Level"

## [21] "X-Spam-Status" "Received"

## [23] "Received" "Received"

## [25] "Received" "DKIM-Signature"

## [27] "X-Google-DKIM-Signature" "X-Gm-Message-State"

## [29] "X-Received" "MIME-Version"

## [31] "References" "In-Reply-To"

## [33] "From" "Date"

## [35] "Message-ID" "To"

## [37] "X-Tag-Only" "X-Filter-Node"

## [39] "X-Spam-Level" "X-Spam-Status"

## [41] "X-Spam-Flag" "Content-Disposition"

## [43] "Subject" "X-BeenThere"

## [45] "X-Mailman-Version" "Precedence"

## [47] "List-Id" "List-Unsubscribe"

## [49] "List-Archive" "List-Post"

## [51] "List-Help" "List-Subscribe"

## [53] "Content-Type" "Content-Transfer-Encoding"

## [55] "Errors-To" "Sender"

The mbox object does provide a get() method to retrieve header values so we'll go that route to build our data frame but we'll make yet-another helper since doing something like msg1$get("this header does not exist") will return NULL just like list(a=1)$b would. We'll actually make two new helpers since we want to be able to safely work with the payload content and that means ensuring it's in UTF-8 encoding (mail systems are horribly diverse beasts and the R community is international and, remember, we're using R mailing list messages):

# execute an object's get() method and return a character string or NA if no value was present for the key

get\_chr <- function(.x, .y) { as.character(.x[["get"]](.y)) %|0|% NA\_character\_ }

# get the object's value as a valid UTF-8 string

utf8\_decode <- function(.x) { .x[["decode"]]("utf-8", "ignore") %|0|% NA\_character\_ }

We're also doing this because I get really tired of using the $ syntax.

We also want the message content or payload. Modern mail messages can be really complex structures with many [multiple part entities](https://tools.ietf.org/html/rfc1521#page-28). To put it a different way, there may be HTML, RTF and plaintext versions of a message all in the same envelope. We want the plaintext ones so we'll have to iterate through any multipart messages to (hopefully) get to a plaintext version. Since this post is already pretty long and we ignored errors in the tm portion, I'll refrain from including any error handling code here as well.

map\_df(1:py\_len(mbox), ~{

m <- mbox$get(.x-1) # python uses 0-index lists

list(

date = as.POSIXct(get\_chr(m, "date"), format = "%a, %e %b %Y %H:%M:%S %z"),

from = get\_chr(m, "from"),

to = get\_chr(m, "to"),

subj = get\_chr(m, "subject"),

spam\_score = get\_chr(m, "X-Spam-Score")

) -> mdf

content\_type <- m$get\_content\_maintype() %|0|% NA\_character\_

if (content\_type[1] == "text") { # we don't want images

while (m$is\_multipart()) m <- m$get\_payload()[[1]] # cycle through until we get to something we can use

mtmp <- m$get\_payload(decode = TRUE) # get the message text

mdf$body <- utf8\_decode(mtmp) # make it safe to use

}

mdf

}) -> mbox\_df

glimpse(mbox\_df)

## Observations: 198

## Variables: 7

## $ date 2018-08-01 11:01:17, 2018-08-01 09:09:18, 20...

## $ from "jim holtman ", "PIKAL Pe...

## $ to "diego.avesani@gmail.com, R mailing list "Re: [R] read txt file - date - no space", "R...

## $ spam\_score "-3.631", "-3.533", "-3.631", "-3.631", "-3.5...

## $ content\_type "text", "text", "text", "text", "text", "text...

## $ body "Try this:\n\n library(lubridate)\n library...

**FIN**

By now, you've likely figured out this post really had nothing to do with reading mbox files. I mean, *it did* — and this was a task I had to do this week — but the real goal was to use a fairly basic task to help R folks edge a bit closer to becoming more friendly with Python in R. There hundreds of thousands of [Python packages](https://pypi.org/) out there and, while I'm one to wax poetic about having R or C[++]-backed R-native packages — and am wont to point out Python's egregiously prolific flaws — sometimes you just need to get something done quickly and wish to avoid reinventing the wheel. The reticulate package makes that eminently possible.