We can normally do a lot of automated reporting with R. Being able to easily and reliably send emails is a high priority.

There is already a selection of packages for sending email from R:

* {mailR}
* {gmailr}
* {blastula}
* {blatr} (Windows)
* {mail} and
* {sendmailR}.

We’ve had the most experience with the first two, both of which are really solid packages. However, {gmailr} uses the Google Mail API so it doesn’t work with all SMTP servers and {mailR} has a dependency on {rJava} which can be a bit of a hurdle for deploying in some environments.

We use {emayili} with the following two design goals: works with all SMTP servers and has few (or easily satisfied) dependencies.

**Protocols**

A web browser uses HTTP ([Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)) to communicate with a web server. Similarly, an email client (like Outlook or Thunderbird) uses SMTP ([Simple Mail Transfer Protocol](https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol)) as the communication protocol to send emails.

Neither HTTP nor SMTP is a secure protocol. This means that data are transferred over the network without encryption. Sensitive information (like passwords) is sent in plain text, which means that it can be intercepted and used for nefarious purposes. However, both HTTP and SMTP can be wrapped in a secure layer (using either [TLS](https://en.wikipedia.org/wiki/Transport_Layer_Security) or SSL) which renders communication secure.

I needed to find a way to get R to speak SMTP, with the option of securing that communication as well.

**What is curl?**

curl is a command line tool for transferring data. You’re most likely to have used it for downloading data from a URL. But curl is much more versatile than that: it speaks a wide variety of protocols and can be used for a bunch of different communication tasks.

In addition to being a command line tool, there’s also an associated library which allows curl to be easily integrated into other software systems.

**Can you send emails with curl?**

It turns out that curl speaks SMTP, so you can use it to send emails.

Let’s see how that works. We’re going to need a username and password to authenticate on the SMTP server. We’ll store the password in an environment variable.

export PASSWD="bd40ef6d4a9413de9c1318a65cbae5d7"

We also need to stash the contents of the message in a file, which we’ll call mail.txt.

From: "Bob"

To: "Alice"

Subject: Bazinga!

Hi Alice,

I’m sending this email using curl. Ain't I l33t?

- Bob.

Now that we have everything set up we can send the email. We’re going to use Google’s server, which listens for secure connections on ports 465 and 587.

*# Using SMTP on port 587.*

curl --mail-from "bob@gmail.com" **\**

--mail-rcpt "alice@yahoo.com" **\**

--user bob@gmail.com:$PASSWD **\**

--upload-file mail.txt **\**

--ssl smtp://smtp.gmail.com:587

*# Using SMTPS on port 465.*

curl --mail-from "bob@gmail.com" **\**

--mail-rcpt "alice@yahoo.com" **\**

--user bob@gmail.com:$PASSWD **\**

--upload-file mail.txt **\**

--ssl smtps://smtp.gmail.com:465 --ssl-reqd

The pertinent command line arguments are:

* --mail-from – email address for the sender
* --mail-rcpt – email address for the recipient (can give this option multiple times)
* --user – username and password for authentication on the SMTP server
* --upload-file – the email data (a text file consisting of header and body encoded according to [RFC 5322](https://tools.ietf.org/html/rfc5322.html))
* --ssl – attempt to use secure protocol and
* --ssl-reqd – always use secure protocol.

The first two arguments are mandatory: you must tell curl who is sending the email and to whom it is being delivered.

It works! Either of the commands above will dispatch an email. So we can send mail from the command line using curl. What about doing it from within R?

**Can you use curl from within R?**

Yes! There’s a curl package for R. At the moment you need to install a development version to get SMTP working though.

Let’s send an email.

**library**(curl)

h <- new\_handle(

verbose = **TRUE**,

username = "bob@gmail.com",

password = "bd40ef6d4a9413de9c1318a65cbae5d7",

mail\_from = "bob@gmail.com",

mail\_rcpt = "alice@yahoo.com",

use\_ssl = 1

)

con <- file("mail.txt", open = "rb")

handle\_setopt(h, readfunction = **function**(nbytes, **...**) {

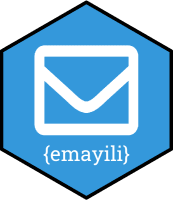
readBin(con, raw(), nbytes)

}, upload = **TRUE**)

curl\_fetch\_memory("smtp://smtp.gmail.com:587", handle = h)

close(con)

Success! Okay, so it seems like we can use the {curl} package to send emails from within R. Next step: wrap that up in a package.



**Implementation**

At the core of {emayili} is are two classes:

* envelope (used to create a message) and
* server (used to communicate with a SMTP server).

First create an email object.

**library**(emayili)

**library**(dplyr)

email <- envelope() %>%

from("alice@yahoo.com") %>%

to("bob@google.com") %>%

subject("This is a plain text message!") %>%

body("Hello!")

You can add Cc, Bcc and Reply-To header fields using the cc(), bcc() and reply() methods. Files can be attached using the attachment() method.

Now create an object to communicate with the server.

smtp <- server(host = "smtp.gmail.com",

port = 465,

username = "bob@gmail.com",

password = "bd40ef6d4a9413de9c1318a65cbae5d7")

Finally send the message.

smtp(email, verbose = **TRUE**)

**Feedback**

We’ve tested this on a couple of different SMTP servers and encountered a few hurdles. We’d appreciate feedback and are very happy to help work through any issues that you encounter.