Language engine for including data in Rmarkdown

David M. Kaplan

3/27/2020

Overview

This document contains a proof of concept for adding data engines to Rmarkdown that would allow placing data directly inside Rmarkdown documents to create completely self-contained Rmarkdown documents. The implementation is based on the idea that `data` chunks will contain the contents of the data files, potentially encoded as text using some encoding method.

Though many types of data chunks are imaginable, I have currently implemented three: `text`, `csv`, `RDS` and a generic type where the `format` chunk option should be a loader function for loading in a file containing the decoded chunk data. For binary data chunks (e.g., `RDS`), the data must be encoded as text before inclusion in the chunk. Two different encodings are currently implemented: `base64` and `gpg`. If the `encoding` chunk option is not supplied, then `base64` is assumed, except for `text` and `csv` for which no encoding (i.e., `asis`) is assumed.

I have included helper functions for simplifying encoding and decoding content using the methods described above. Encoding and decoding operations rely on the base64enc and gpg R packages.

The implementation is quite simple, but it works well. One potential drawback is that this could encourage creating Rmarkdown documents with large datasets inside that are unreadable. This could be prevented or discouraged by limiting the size of `data` chunks, perhaps with an chunk option to allow the user to violate size limits if they really want to.

Implementation of language engine

```r
# Helper function to decode encoded text
# Returns name of file where decoded text was saved
data_decode = function(code,encoding,options,file=tempfile()) {
  switch(
    encoding,
    asis = writeLines(code,file),
    base64 = writeBin(base64enc::base64decode(code),file),
    gpg = {
      tf = tempfile()
      writeLines(code,tf)
      on.exit(file.remove(tf))
      writeBin(gpg::gpg_decrypt(tf,as_text=FALSE),file)
    },
    stop("Uknown encoding: ",encoding)
  )

  return(file)
```

```r
}

# Helper function to encode (typically) binary files for inclusion in data chunks
# Silently returns encoded text. Encoded text is also spit out to console for
# copy-n-paste to Rmarkdown document. This can be surpressed by saving encoded text
# to a file using output argument
data_encode = function(file,encoding,options=list(base64.linewidth=64),output=NULL) {
  code = switch(
    encoding,
    asis = readLines(file),
    base64 = base64enc::base64encode(file,linewidth=options$base64.linewidth),
    gpg = {
      if (is.null(options$receiver))
        stop("Missing GPG receiver. See ?gpg::gpg_encrypt for details.")
      gpg::gpg_encrypt(file,options$receiver,options$signer)
    },
    stop("Uknown encoding: ",encoding)
  )

  if(is.null(output)) {
    cat(code,sep="\n")
  } else {
    writeLines(code,output)
  }

  invisible(code)
}

# Data engine itself
eng_data = function(options) {
  vn = options$output.var
  if (is.null(vn))
    stop("output.var must be supplied in data chunks.")

  format = options$format
  if (is.null(format))
    format = 'text'

  encoding = options$encoding
  if (is.null(encoding)) {
    encoding = switch(
      format,
      text = 'asis',
      csv = 'asis',
      RDS = 'base64',
      'base64'
    )
  }

  encoding.ops = options$encoding.ops
  if (is.null(encoding.ops))
    encoding.ops = list()
  if (!is.list(encoding.ops))
```

```
    stop("encoding.ops should be a list. Got object of class ",class(encoding.ops)[1])

  format.ops = options$format.ops
  if (is.null(format.ops))
    format.ops = list()
  if (!is.list(format.ops))
    stop("format.ops should be a list. Got object of class ",class(format.ops)[1])

  # If format='text', just spit back code, possibly after splitting using a separator
  if (is.character(format) && format=='text') {
    code = options$code
    if (!is.null(format.ops$sep))
      code = do.call(c,strsplit(code,format.ops$sep))

    assign(vn, code, envir = knitr::knit_global())
    return(knitr::engine_output(options,options$code,''))
  }

  # In all other cases, decode data first
  fn = data_decode(options$code,encoding,options=encoding.ops)
  on.exit(file.remove(fn))

  # If character format given, convert to a loader function
  if (is.character(format)) {
    format = switch(
      format,
      csv = read.csv,
      RDS = readRDS,
      get(format,mode="function") # Attempt format as function name
    )
  }

  # Load data
  data = do.call(format,c(file=fn,format.ops))

  # Assign to output.var and exit
  assign(vn, data, envir = knitr::knit_global())
  knitr::engine_output(options,options$code,'')
}

# Add to knitr's list of data engines
knitr::knit_engines$set(data=eng_data)
```

Test of text chunk

Note that `format="text"` is optional as data chunks default to `text` if no format is specified.

```
This is a text.
It has two lines.
```

```
t0
```

```
## [1] "This is a text."   "It has two lines."
```

With separator

```
This is a text.
It has two lines.
```

```
t00
```

```
## [1] "This"   "is"     "a"      "text."  "It"     "has"    "two"    "lines."
```

For numeric input

```
1,2,3,4,5,6
7,8,9
```

```
t000
```

```
## [1] "1" "2" "3" "4" "5" "6" "7" "8" "9"
```

```
as.numeric(t000)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

Test of CSV chunk

```
id,res
1,a
2,b
3,c
```

```
t1
```

id	res
1	a
2	b
3	c

Test of RDS chunk with base64 encoding

```
H4sIAAAAAAAAA4vgYmBgYGZgZgNiViCTgTUOxE3XgoGBSRjI
YQLid1CakYGFgRNI8yXn5xYkJpfEZ+aVFKcWosmyJCUWpOLF
eMHiEPofSCfIKgcVBjCw/4BKQ9UIIJnFnJiUDDQR2XjWvMTc
1GKoOiaoIGMijJEEMgUAP3IVTdMAAAA=
```

```
names(t2)
```

```
## [1] "a" "b"
```

```
t2
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
## [1] "abc"
```

Test of GPG encrypted RDS chunk

Import private key for decrypting chunk

Normally, the decryption key would not be included in the Rmarkdown document, but I am doing so here so that code works for all. This will import the test key into your GPG key ring. This test key does not have a password, but most real keys would and the keyring management software would ask you for that password when knitting the document.

First the code for generating and exporting the key. Not evaluated, but keeping around for reference:

```r
gpg::gpg_keygen("Test Key","test@test.org")
id = gpg::gpg_list_keys("test@test.org")$id
gpg::gpg_export(id,secret = TRUE)
```

Next the key itself in a `text` chunk.

```
-----BEGIN PGP PRIVATE KEY BLOCK-----

lQVYBF6A3p8BDADcaf7tveXZUpi0IfEpmYrPP8/0SXSh3iBkd5bdTvbq/FwLGIsD
dp/dFqAWS+0BqCIMFAtV63FUOG4kXYpkajdl2QU1Hy0aY9F9K0imc5JUM1SEry5F
CckjzDFp3u4pmmCPWKF2jVnaHzahJfKz9J9qD9BfBSynfyQU2XgsrRqNgiqeNc0i
f0674hpReawnecBwhENKMWL38O1a0tP1IDx9cFI6busii0aIHIYYW6qbv178offy
OOWogstsQ3EJQbPBPkkgVTn8wwGUtoorc/2AonSoz99QC4nMWbBaDUGuE9O32yRv
Q7Pe6bWVBuIeV5ASAfSSEypzNHB576BF6MTy+lJvhfXI41Yu97geQJM0CplJ8xav
xAhIvrKjkDoW3zwrZlG54G2TidwEyXoDx7cyRVnCf9tsBCmhEDiKvzlg2IE9Fo65
+LWrD12qCKi7cu4XE28q4zy7S4adhUCBcuflZ8wKMVvbZRXvqnAHBAK8gQxMqHMc
EjWAb7rvmN9bkTUAEQEAAQAL/if4vPeGYaGIvhKkuSRvKOIu01O4tIMKUluF6IEX
6eVxgIuulr85CwLAMKX6fO+4+vuvwuKBARth5G+J2ygcrxE0SyJ4FejcQ0hsyg8N
lHLaoDAzyLNSc/ye8jMd75jx2yMD0rw6JBpPYMvWou4JpcNJPOOOf6ucfgGd8pI/
jjotaecpHuJgLfoapeUyqIq8JK8C/WT+EdGfCpw7YObqQq4I6ZCZPuETbKMwcQ0H
yqfWC7bK9Lk/MvbdSWDH1j70f/t1KaUEBZ2z5xTALqxaFgbwXh+7FybzV+09Sxsn
l5deeubEQXwkbPthapjRpvRo197tJRHLJ8wQVCwag39ip5cvuWQIsej3qILKTepz
VBdgZa4hIyLX8uUCAtLrVYwvWzV1oWxPLAkXJ6KPCzB0jQb7q7UUyrBOUaavdnt2
aWBz5EuXPTaMqnzWqEKIazcXqiCSNjIEv7HWcU734IGUazYper3poYgOWYYIdUes
+xbdWP/j6313N3u4a9BSd3PMvQYA4CLwr+gBfX+dybX3jq3ldB3HJS/Lv90e64rh
BarRu+ByyEO5BcVJZ+ZEUOcBjF/pvG1qI9mfqBuZX/e2aW1lmMsxcXNlWRu5b5vE
geoRwqPMNIo4JIo2hByHZeEPQLcYW/QRy5xkoNbl+udPuS3PMEUnfnPeQKursY71
ao7ZoOTUeFRemEgkvxZpFXfT+IMs9DGI/Wi6POOChSJ/Cu/QixgK0eJFUroNCyvl
bW+xyOGSB325wkyOM5xIny681KtvBgD7v5V6nOP2UucxZYU5hhdWaaTf5aF83vtE
o88gSU5NRO1/wPFb+AFP3fw8TNtrvRlA/OakwjL+GbfhioAJ4mtPbdGUojFIAU6X
czMHbaYyNwZTMImBW9uc2gDqta8O1HiSwC7fXnTxVoSz3E/TD6dbAnFyf1FYNntJ
PLKS9H82idCqO0nrU3LtdKJx9VHJ6wLOT16D6zZAdgNB0wK9dzStayfIqQzN/FAz
O1u0ehX4SDRCxxgukdR4ZyeZJfdmC5sF+wZ/2mW4Tp7v3kutNAytk4JtMvLIhe2r
BQkYw5eUFMq7tUqXgsXMjA0pVplUSosZknCIpoyoEU7rvS9BF9xdcpRixU5kxeYY
knQg5jtb+vx3Stpp0vbuvFFaGgEJhNP6Tg3al7gBCOwEEAJmSTko4cyf1e45pIMF
+jGbIeozSjeKPWjdJCr4q05tvKgsiAe7BulgUlNhS6Ty5JyQHsiM/WZTPko2BsN2
8Apa/nuOvYwRwFLGGXVVWV3jQroPI9Hbft9ctBhUZXN0IEtleSA8dGVzdEB0ZXN0
Lm9yZz6JAdQEEwEKAD4WIQTvl6O3A7Tx/z8NeR/qzEhnRW4g7QUCXoDenwIbAwUJ
A8JnAAULCQgHAgYVCgkICwIEFgIDAQIeAQIXgAAKCRDqzEhnRW4g7WxrC/94WT6J
HEEgyb9Bskm2ik+c/qUW8w7JgizYRi6jqi8+qiIesh99MZ/XPm5mgMTIvKr0z/IG
xaU+RKYFF5DqsAc4obg/ZmClOSY9FgDWlMEm7hEqourQxfJZXGWRNcU6DTr2tC/K
GpTNkhR8O2LnjUePeVJU5MMuJ8eyQV+NgGhwXTIcPA6ERwHIC1n24N3QDFNoijcc
pTi5p9+N33w8fBC5ZMeZwrWI6mCJjEWVbxG2zcsIJ2t7htWRM7W1rKi5lHRpQdn/
cd9WtbdDFj7ywGPnjMB2vxYVJreENGbE/LZIZPaJKJHPReWQ+GBSGkyY7nrT32SP
```

```
R+qj5gO0Bez7F+61EDU+SXP9PJ8fyTGtUWfTsgz+fTj2TDn39y0tL1wuSciEOAjD
uia+L5qiKE9GK6mBQv78yfzZ/ZOEdJn9ZNRWs8kvs/aG9BygYMdJM5T4vvk2DcWd
mO61EGTg/AVUFpMuTon9tb+RCIFfVjSzat8LWcf4Me2nJeFZu+lW/lCmxkedBVgE
XoDenwEMANPff6PrZirginP4HNK7g3ANmB3bDKCI1msAQspXMzvhtMc0Hn8DpM+r
wPUuoOo4hnYwkGHSNZ4dulrtW99mlzQWcFwDuOsvPAqc/OuEIEoOBBvc5HcpNk4d
z94Vno+Dq904VnlStf6DXpGbBFZkZBoC4XVwFUSoEjD1i967ckjFUhOxE5ynlcMb
8mpS65iml4JFd572bcuo9exJ1g7IhdgFIFoDDD2eJkxEhmEHNiVd8B9/j1GHxDCq
v/DOHNbgKuFk8WJUMYvupdqA30wAc5Ujnf+nURfNejgZTOiGXm5FZBrw/dha7yTP
/mlnNFMBKUEBrxYyPo2JVSsYfPf1WzLL1dmv8JPC5fyEKYhEC+zBvlytRWqkZV88
DumgVEdhEnnMEVlofyF8KoVMmWYA9w/FUUKiNymZlK1PEGecqliEhXh+KE03ncHh
AyEoOZcdh5sSxUW5fNsQb+tp0fqFBs7Yye432w6ID3ZIONrnWrQ6MewWwxeAGMam
xO3jgyMlCwARAQABAAv9EJ0e8iicS1JuKOfUwsWHafr26ahqlhAE2EEd+6XY06JA
PbqdhZIwk0RBjjhIz/T8vjnSqIkGQU7NdSHVqW/u/VuhFeYI0xBSIfbrckBbE9Z+
V/z7QUjPBFMcIKsLUu+dQ2yOg1b0BHAis0I3ldqrasq9CStvz4FqY8JtZFrIfGJU
rEyfYBJYEQOY/7Ne3Ap8KO/vkFx8gZLPLecgTOp2bFkCj2xbwl0rXaGl8+fP3CBA
mweyok8GGFbbVDagKE1NiukpEVzHsoMyMfPkxdIMLSj0F2GzQSnhyhyGomNstuTT
EC/i3/u7M9TRvLkpNTP3I6z5VNjayrp0NBs0z3sb1wNzrACELWbTtb/Lo5BVVD9Y
mOMQtDi8+SKzTHci2AdpvewxnhO4IiS/aXYYGcPwmEX4YdlZeV0J5mRXNsvWxYZk
HHFkbfgUkiFSFOmb9uyPD0NMldJoLXbv9+LFiU1okglietVcKK7Fyt5xCKcxbtO8
kdYJTuWonsWeyC8tz1WBBgDcq6doxs3aFSVeLcZ0//WHif+iBYlLFoexmw4irx8e
LnZilDJ5i4mwcu6Q5qxao3UEyeUC7ff//Qn846TQMDDRcC3xtrbqAqVyYBE7u9EI
OMyyCfosk8nNmVBpNdnsFm76lUyG8GiuT6b0j8BiQTRPmH4Xlh3pSiihyuTJIVhX
Y663wV8EwT9IRnYCoVqw9s5qZqJGkI4rxnABuyJui4BpmkrLry70t1xb6MdX2BPD
eK5u0YJ24AmxPW5YGvXnO0sGAPXLRfarrI9IgSz28+QpfYttOIbjp3n3AxB3ImHo
oK+CLsc1vHtsdEV8hElWo9k5EqcdlhPBbeC6IILFqT69Ldx8jK85hxR0bYs2NVLC
qyWo1T3bovPePCEenN4++VPBtVBkEt51MByNIKwC3Bw0zvHcygLcHE3iXRQ40dhq
AZWrPlOqwnC8x9+UqZoWCp/JRWD5qBjD6EPVAxwbtcUdjDOhZ1y51xbUaX59Vlul
BGLse/0Q47m71HrF+d9rGUnlQQYAkDQsdbzijmB/tVzcRXJWbZVgjwLciofxVpoM
TEYyw8+oSYDI1L3Dikejp3XymVr+9pKGmPZjLqL9Q01J9epeHt5wgLjuWTXtkVLW
kbnt7vTy257BIsHGDwiJzMI7PujTlQ4B1ZTPz2WyUJ7gn1f+J9wYpNOr7qeE2pg6
cOeiPQmT5h88jWTUH/eAJ0nAWx46kwgQY4uZz7xsFtCcwQgqVe9bD5MNv/bBUdPW
RkF8ZbRCPRk4Vl2DYM/rXC2VGCFZ6OeJAbYEGAEKACAWIQTvl6O3A7Tx/z8NeR/q
zEhnRW4g7QUCXoDenwIbDAAKCRDqzEhnRW4g7ZayC/954y+kfmjtIzSRDBRpOo2s
npOOwy7RLdOdWvab6jVecyqYsDyd/fiCXVKxALOVR31WTef00iFSLHQactwFxQyJ
zY6YO8tGkvYEXXYJR5O5MNzjlhNMndBqGIbKe9tA2BFLDD/6mmvMD/i9k+IhHzFT
NhoczB5rE9oaApMZhAj9u9Uv2zy0osfcOPcy+RN9b2noodVS/7Ei2BjWl+V/MGqa
I8oBM/ETIW/jcq+OuE8oSqoByFtFHh1DgOzOFugCWApOmAjLQwQCmDiYYtKN1GWq
l1E+txLud78ZBsJQL/78MXO9V2T2dCbcIA0vOfACuoPApfu6seREOSLeImgoRg+8
7aX6HtiRXRjExDS26YNbGYzAvVTl3Zy1VptXOMwkh5CcIgtTcDv32pLWC3xvNydG
P4xDMM+BVuDi6QTcFfbPtqYbuuT4OFyyaSzee0oWxvKoX2pL81VnMwvb7Uy47Dxf
Ng9Af4cf3nf9UzesAVbSy1gtvlZIyX0HwtZNVLNJSS4=
=C6UF
-----END PGP PRIVATE KEY BLOCK-----
```

```r
tf = tempfile()
writeLines(key,tf)
gpg::gpg_import(tf)
```

```
##      found   imported    secrets signatures    revoked
##          3          1          2          0          0
```

```r
file.remove(tf)
```

```
## [1] TRUE
```

Data chunk

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v2

hQGMA9TPonHna5j3AQwAqNofDHURDA3k4I64cahxxO+Nv88ppT0rJ4+9Bbvr1VcC
7fXfudpfb7fj0SRZWagquyBnvu4vAVFsyzvBrvcg4WqW+cgM/m11rTjM/idkoZcl
DuLrkmHnRCuwVftw9hyO+/ghGKW95CQljqksq5rL+ZVjyF3PFXBRMGJXoYo6h2qe
AmJlWLHWuIV8BHtyFWhUum+VUXOTDuixMt1AeRwaohQgY1HPpQp+w5Xv+OGK4Evg
7dn0xg/USWtYDGOWnP7xPXMQIYP77XHdhlzmuinPhT3h37cQciswKryWbSdq6Mr+
a7g5h/zjQ1f+DbhMj0zfmOW2NKiF/rCoVOclNGzzLYBZ9RgJOZvvNOslRnO4E24y
tcds64cGT6z3y7mrYrDQpnHrOOGCM/Sk/p4gqiiSB2oBHq5ZZqbxqINWVjZRuZD+
iKWdP1F4WItPk6SAXM5OlcSTQjldpRtJRaasZQR5Om/sb/JiuU0dOKSHc2UVvvVN
DfuodG7HCFdWOXgzStHU0r8By5BAyIu+ETLys9Dkwr3aPj+0rbZUb5k9rY+XPYmZ
VobFt/kpqzBWtTr4nJqUpFtDTXQSZLceri7bfUEcFGCtYpkExh3RaKagjn7lHLOk
Rig3esmXKQhL9kRw5UhX4ti71ntAswEs4LCqPrWLJ43EEKbokmdFdE6R35tsqniq
2tADnUazcqWXMiLqcbxo6TwW1eqmLuLwLHyM7WPrinizgS+sEED1uS+xRVrd1ruJ
oH9QuxSp8POmihI53+AbVA==
=WRy/
-----END PGP MESSAGE-----
```

```r
names(t3)
```

```
## [1] "a" "b"
```

```r
t3
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
## [1] "abc"
```

Encoding a binary file for incorporation in Rmarkdown

Below I demonstrate how to use the helper function `data_encode` to encode binary files for insertion in `data` chunks.

```r
saveRDS(t2,"test.RDS")
```

base64

```r
data_encode("test.RDS","base64")
```

```
## H4sIAAAAAAAAA4vgYmBgYGZgZgNiViCTgTU0xE3XgoGBSRjIYQLid1CakYGFgRNI
## 8yXn5xYkJpfEZ+aVFKcWosmyJCUWp0LFeMHiEPofSCfIKgcVBjCw/4BKQ9UIIJnF
## nJiUDDQR2XjWvMTc1GKoOiaoIGMijJEEMgUAP3IVTdMAAAA=
```

GPG

```r
id = gpg::gpg_list_keys("test@test.org")$id
data_encode("test.RDS","gpg",options=list(receiver=id))
```

```
## -----BEGIN PGP MESSAGE-----
## Version: GnuPG v2
##
## hQGMA9TPonHna5j3AQwAklgn76V+wB1TztaQt24mICEXPalKkLDCbb6mE8AnrJM3
## rTnOv8/5QTuJLf+YHmbg5L23sWen0gQHcQiqCrfASWb/3wF91E9HTbs11FzuZ7z+
## 7eGamme7+ok4MbJgmnxzFdpRO2QntatcHw5WzW06B/Y4rlC3KZvzV6/bgAUd3iRq
## F6xU9NYITndDasVSx14deOSE61fPTudaMNkx+WUrmlv5AHWR7WNeQaPHwjT73h8d
## ETotJozKsI6ekiufbACcBk9EiEBR+d6rqPSAbwcbLL8Cnf1icEm0h5QjF+X1XwgA
## BUQREniDPk/uJ1OBK9cC16rOCiqIIXHWAj3gLvWQvvsMu7p8esf+pVfhn1gGBKpX
## PKxgDisS93jQi8qIQMHsiU6o5MrYJnpt6n3P/XUXDCNWAGaue7CFnt3X/uDqxGfG
## IrCAVi8Cmynd5zeFol2ipFeeXFJCvJk+RVOGyHbC7xK42QLUi6FgupAlONfn4RZE
## rjy+O4qRe4/ux+m4VDJF0r8BGxz8rq5VaHLUmBT5+OTD95dC38QhWQrnWZldlS/+
## XNykFS4crumO+bjkpbL9IQrs5Nb4SMaZse25u+XExmvOULMGgUtSXdlgNa41mXTb
## 4Ej3S2Wc0Or6GmCUC/50ke8E8vvW79UAyJ4OmMOl69xkaEGBt9qHLOhgV+j4SCr9
## W2MkAzcC5Dam8CmmEJ2rIvX6QAWAgCY03cIQN1DTqfwjKoa3fKkUTHpe6Tua72x9
## aTYJg5rvvMTgxDlSKGSR0w==
## =mg0Q
## -----END PGP MESSAGE-----
```

GPG saving to file

```r
id = gpg::gpg_list_keys("test@test.org")$id
data_encode("test.RDS","gpg",options=list(receiver=id),output="test.gpg")
cat(readLines("test.gpg"),sep="\n")
```

```
## -----BEGIN PGP MESSAGE-----
## Version: GnuPG v2
##
## hQGMA9TPonHna5j3AQv+JgHZMnA3MFf06uRYTumXF3DUM32RnEEzCgelR9CM930B
## vh/+lJe8unb3xaxWXNM0n/I8dmzicYlSuueZwrP6MW411Q6YHsnJ3BvLpyECI2Qn
## vr8+FsZ35uaRE43CEgrA5bANqrl8CnT1FWbU32VAudJCdAoofCmo+CZefSi2d+0j
## qT5WTlupveHAw5I/f1tUicTMHg79VpxPK0BRjJWxVeU8M50lIrcsVn6khTVMnIoW
## 0+2e3o44XBOi8OdIbXxMhj31D9iC6ybRpmwv6E+VcqTiBMK6+PzxIO2magyxur0h
## iXZXKGj5Eap9rZ8EDNxyoW0ns0O7AdkzstG0BZnQUTK6FPMgUlCNdn1kK7U1GIHp
## XoabR+I4IgonOVlPw/18j5vdKS3gHd9P79SzXrINzj/5gYxoeW2oBtnNNQgH+flG
## L1oFL3g2WmpS0sw523ArZQq0h81vV7XdfL37K5j0jqJ661DzVb3bF/NGk4/BBrnC
## lA2NPcu1dfcQN5Wm7z5d0r8Bil1CD467TBw42impfs2RQpFsn9O0j5mHKuR0TjOL
## 1sWAAos6tb6EsEPe8xjnLTU3zvdCkfbiqmv0tu0Qk7BPDOkS/2SQ3iY1N4UEM1s1
## zMoh7h2hqWfv1AcSln55OIt0hv7mGdONPvwaT6UvVPnOY3RKkChB+gA6CIb89/3o
## XKcSYrtDFLES8Qggc6Q3sE46WOD5k3dB2NNw6xlm9/SBJpxQ3B7SAZJQT1ifldrJ
## QOeHQH23ARH4qD9m8WZ+gQ==
## =bw8q
## -----END PGP MESSAGE-----
```

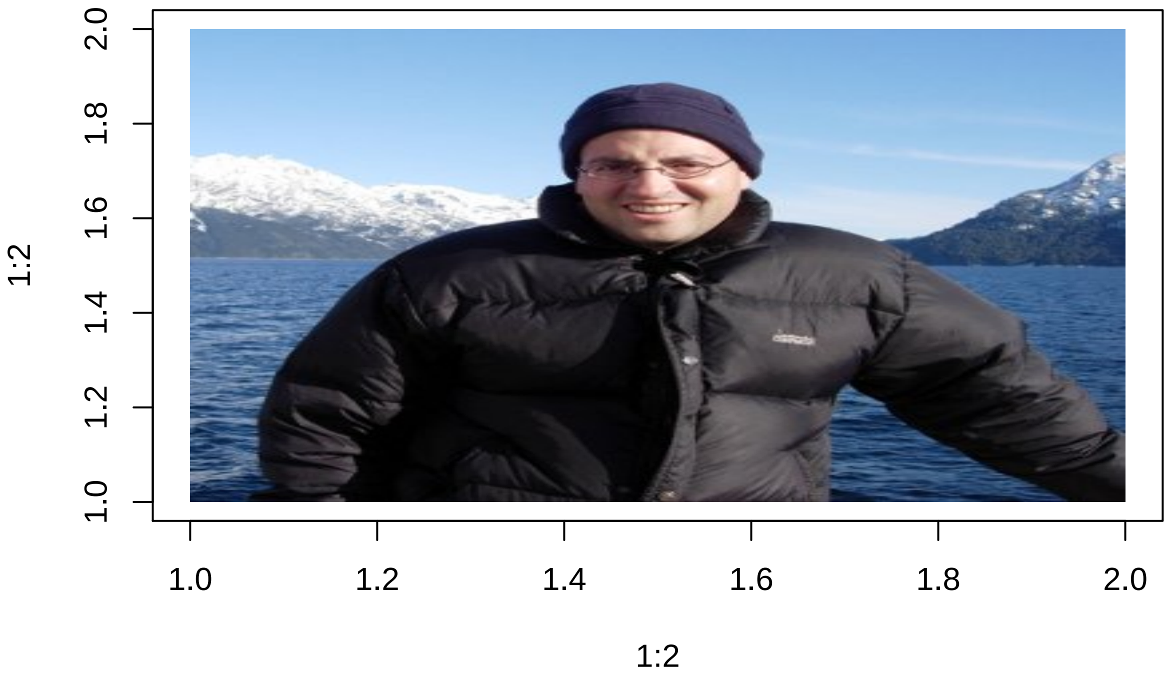
```r
file.remove("test.gpg")
```

```
## [1] TRUE
```

Test of arbitrary binary format with loader function

Note that `echo=FALSE` is quite important here. *Perhaps I should force this on all data chunks?*

```r
plot(1:2,1:2,type="n")
rasterImage(img,1,1,2,2)
```



Remove imported key

I am cleaning up by removing the imported GPG key. The keyring management software may ask you about this before doing it.

```r
id = gpg::gpg_list_keys("test@test.org")$id
gpg::gpg_delete(id,secret=TRUE)
```

```
## [1] "EACC4867456E20ED"
```