An simple R interface
to Google Documents

Table of Contents

This illustrates how to use the code that we put together quite rapidly to communicate with the Google Documents manager. It uses the XML and RCurl packages and illustrates that it is relatively quick and easy to use their primitives to interact with Web services.

The first thing to do is login using *getGoogleAuth*() . You specify your login name for Google, e.g. dtemplelang@gmail.com, and your password for that account.

```R
auth = getGoogleAuth("dtemplelang@gmail.com", gpasswd)
```
We can put this "permanently" into a Curl handle so that we don't need to specify it in all the calls with

```R
con = getGoogleDocsConnection(auth)
```
We now just pass **con** as the value for the *curl* parameter in calls to the other functions. In the future, we might provide an interface that avoids this such as

```R
con$getDocs()
 names(con)
```
and where the **con** value is implicitly provided.

Note

We do have to specify auth when calling *uploadDoc*() .

Having logged in, we can find out about the existing documents.

```R
docs = getDocs(con)
```
This is a list of R descriptions from the XML contents.

```R
names(docs)

[1] "my foo1"
[2] "my foo1"
[3] "my foo1"
[4] "my foo"
```

```
[5] "another bob"
[6] "\"~/foo.csv\""
[7] "SampleDoc"
[8] "Copy of This is a sample document that we are..."
[9] "bob"
```

R

```
names(docs[[1]])

 [1] "id"         "published"  "updated"     "category"    "title"
 [6] "content"    "alternate"  "self"        "edit"        "edit-media"
[11] "author"     "feedLink"
```

R

```
docs[[1]]

$id
[1] "http://docs.google.com/feeds/documents/private/full/document%3Adfwhmfk3_6c26h

$published
[1] "2008-09-24 14:02:56 PDT"

$updated
[1] "2008-09-24 14:02:56 PDT"

$category
                                              scheme
        "http://schemas.google.com/g/2005#kind"
                                                term
"http://schemas.google.com/docs/2007#document"
                                               label
                                        "document"

$title
[1] "my foo1"

$content



"http://docs.google.com/feeds/download/documents/RawDocContents?action=fetch&justB

$alternate
                                                 rel
                                        "alternate"
                                                type
                                        "text/html"
                                                href
```

```
"http://docs.google.com/Doc?id=dfwhmfk3_6c26hc6gr"

$self
                                                                          re
                                                                       "self
                                                                         typ
                                                   "application/atom+xml
                                                                         hre
"http://docs.google.com/feeds/documents/private/full/document%3Adfwhmfk3_6c26hc6gr

$edit



                                                    "application

"http://docs.google.com/feeds/documents/private/full/document%3Adfwhmfk3_6c26hc6gr

$`edit-media`

                                                              "edit-

                                                               "text

"http://docs.google.com/feeds/media/private/full/document%3Adfwhmfk3_6c26hc6gr/fli

$author
dtemplelang@gmail.com
         "dtemplelang"

$feedLink
                                                                          rel
                            "http://schemas.google.com/acl/2007#accessControlList"
                                                                         href
"http://docs.google.com/feeds/acl/private/full/document%3Adfwhmfk3_6c26hc6gr"

attr(,"class")
[1] "GoogleDocument"
```

We can turn these into a data frame with

```R
as(docs, "data.frame")
```

We can fetch a document with

```R
getDocContent(docs[["SampleDoc"]], con)
```
or directly by the name of the document

```
                                                                              R
getDocContent("SampleDoc", con)
```

We can remove a document with

```
                                                                              R
deleteDoc("my foo1", con)
```
or
```
                                                                              R
deleteDoc(docs[[1]], con)
```
You can check what files remain with

```
                                                                              R
names(getDocs(con))
```

Finally, to upload a file from your system to your Google Documents account, we use *uploadDoc*() . For this function, you currently have specify a value for the *auth* parameter. This is the value returned by *getGoogleAuth*() earlier on.

uploadDoc() you can supply a file name or the actual content to upload. If the value of *content* matches a file name, then we read the contents of that file and upload that string. If this is a binary file, you should read the contents yourself and pass those as the value for *content*.

We try to determine the type of document (e.g. a spreadsheet, a Word document, a CSV file) from the extension of the filename (using *findType*() and matching the extension - MIME type table from Google's documentation). If the extension doesn't match or if you are specifying the content directly, you should specify a vale for the *type* parameter. This can either be the MIME type (or an initial part thereof), e.g "text/html" or "text/tab", or you can provide the corresponding extension, e.g. "htm" or "tab".

The *name* parameter allows you to specify a name to be used as the title of the document in the Google Documents manager panel.

Note that Google cannot convert all types of documents and does not necessarily even handle "rich" CSV files.

So let's upload a CSV file with the contents

```
1, 2, 3
4, 5, 6
```

We'll first upload the contents directly

```
                                                                              R
x = "1, 2, 3\n4, 5, 6\n"
uploadDoc(x, auth, name = "direct csv", type = "csv")
```
If we put the contents in the file /tmp/foo.csv, then we can upload this as

```
                                                                              R
uploadDoc("/tmp/foo.csv", auth)
```
Here *uploadDoc*() can infer the MIME type and the name from the local file name.

Binary files are slightly more complex.

R
```
f = system.file("sampleDocs", "SampleDoc.doc", package = "RGoogleDocs")
uploadDoc(f, auth, type = "doc", binary = TRUE)
```

To upload a spreadsheet

R
```
f = system.file("sampleDocs", "SampleSpreadsheet.xls", package = "RGoogleDocs")
uploadDoc(f, auth, name = basename(f), binary = TRUE)
```

When *binary* is **TRUE**, the *uploadDoc*() function calls *readBinary*() which amounts to calling `readBin(f, "raw", 22016)` as it determines the number of bytes in the file for us. You can work with raw content yourself directly and upload that. This is like uploading the contents as text when there is no associated file but the content was generated from a previous call. For example, let's read the binary file ourselves:

R
```
vec = readBinary(f)
class(vec)
```

Then we can upload it, but again we have to specify the type and any name we want.

R
```
uploadDoc(vec, auth, type = "xls")
```

You can also upload material to a specific folder. We do this by specifying the identity of the folder via *uploadDoc*() 's *folder* parameter. This should be a *GoogleFolder* object retrieved from, for example, a call to *getDocs*() or *listFolder*() . We get the list of "documents"

R
```
docs = getDocs(con)
```

and then can upload a file to say "MyFolder" using

R
```
uploadDoc(x, con, name = "boo", type = "csv", folder = docs$MyFolder)
```

A more convenient version of this is

R
```
f[["boo1", type = 'csv']] = x
```

where we "assign" the value to a name within the folder as if it were a list. Note that we can specify the type and other arguments that are passed on to *uploadDoc*() . Also note that the connection object is obtained from the folder object.

We can also specify the folder by name, e.g.,

R
```
uploadDoc(x, con, name = "boo2", type = "csv", folder = "MyFolder")
```

But be aware that this involves an implicit call to *getDocs*() and so involves an extra "trip to the server".

Finally, if you happen to know the "internal" identity of the folder, i.e. is /feeds/folders/.../ URL, you can specify this. But note that it must be "escaped" using the *I*() function, e.g.,

R
```
uploadDoc(x, con, name = "boo3", type = "csv", folder = I(docs$MyFolder@content["s
```

Tools for working with word processing documents

The word processing documents are just HTML documents. So we can use *htmlParse*() (or *html-TreeParse*()) and the XPath to find what we want. We can get the content or find the nodes of interest and modify them and then upload the resulting document. We have provided some simple functions for accessing elements of a word processing document. These are *comments*(), *images*(), *footnotes*() and *sections*(). Each of these takes either the name of a document and a connection (returned from *getConnection*()) or the parsed HTML document. For example, we can call each of these as either:

```R
comments("Many Parts", con)
doc = htmlParse(getDocContent("Many Parts", con), asText = TRUE, error = function(
comments(doc)
sections(doc)
```

The latter approach avoids retrieving the document and parsing it multiple times.

comments() returns a data frame with a row for each comment and columns giving the text of the comment, the date the comment was last modified (or created?) and the name of the author of the comment.

sections() returns a character vector giving the title of the different sections. The names of the elements of this vector are numbers giving the level of the section. This is taken from the h1, h2, h3, ..., h6 elements in the HTML document.

The *footnotes*() function returns a character vector giving the text of the footnotes. The names are the unique identifiers within the document of these elements.

images() returns the names of the image files referenced within the document. Note that these are not the original names of the image files, but the names as they are stored within the Google documents repository.

Spreadsheets

We have added basic functions for working with spreadsheets. We create a connection for working with spreadsheets rather than word processing documents. We do this by specifying the service as "wise" rather than the default "writely".

```R
sheets.con = getGoogleDocsConnection(getGoogleAuth("dtemplelang@gmail.com", "...",
```

When we call *getDocs*() with this connection, we get back information about spreadsheets only.

```R
a = getDocs(sheets.con)
```

The function *getWorksheets*() is used to obtain a list of objects that identify each of the worksheets within a spreadsheet.

```R
ts = getWorksheets(a$TwoSheets, sheets.con)
  names(ts)
```

getWorksheets() is smart enough to be able to work from the name of the spreadsheet, e.g.

R
```R
ts = getWorksheets("TwoSheets", sheets.con)
```
but it is faster to use the GoogleDocument object returned via *getDocs*() as it avoids an extra request to the Google Docs server.

We can do various things with the spreadsheet and its worksheets. We can query the dimensions and/or the contents of the worksheet or a part of it, we can modify one or more cells, and we can add a worksheet to a spreadsheet. The functions *dim*(), *nrow*() and *ncol*() all work. These report the "declared" dimensions of the worksheet, i.e. how many rows and columns have been allocated. This is often way more than are actually used. The function *getExtent*() tells us about the rectangular region that is actually in use. This returns a 2 x 2 matrix giving the "bounding box" of the effective cells in use. If there is nothing in the first row and column, this would return 2, 2 as the indices of the first cell.

We can convert a worksheet to a matrix or data frame using the regular *as*() function, e.g. as(sheet, "matrix") or as(sheet, "data.frame"). The coercion methods are merely calls to the function *sheetAsMatrix*() which provides more control of the coercion. It allows us to specify how the column names are found (e.g. as the first row of the worksheet, or given in the call as the value of the *header* parameter) and whether to discard "empty" rows and columns. For example,

R
```R
sheetAsMatrix(ts$Sheet1, header = TRUE, as.data.frame = TRUE, trim = TRUE)
```

If we want to access one or more cells, we can convert the entire worksheet into a data frame and then use R's regular subsetting. However, this is potentially expensive in that we have to download the entire worksheet and then process all of the contents. If the worksheet is large and we only want a few values, we are doing a lot of extra work. So we have provided methods for the subsetting operator *[*() that do this more efficiently by retrieving and processing only the specified cells. We can use these on the *GoogleWorksheetRef* objects. For example, suppose we have our sheet with

R
```R
con = getGoogleDocsConnection("me", "my password")
mine4 = getWorksheets("mine4", con)[[1]]
```
then we can get a single cell with

R
```R
mine4[2, 3]
```
We can get multiple cell values, e.g.

R
```R
mine4[1:2, ]
mine4[, 2:3]
mine4[1, ]
mine4[, 3]
```

We can also assign values to one or more cells. Let's start by adding a new worksheet to the spreadsheet mine4:

R
```R
mine4 = getDocs(con)$mine4
sh = addWorksheet(mine4, con, "test")
```
Now we can populate it

R

```
sh[1,1] = 2
sh[2, 1:10] = letters[1:10]
sh[, 11] = letters[1:5]
```

When we omit a dimension, the affected cells range over the extent

Adding Documents

We can use *uploadDoc*() to upload a document or even an R object such as a data frame or matrix which will be converted to a spreadsheet via a CSV upload. We can use *addSpreadsheet*() to create a spreadsheet document with a single empty worksheet of specified dimensions. This is a simple wrapper for *uploadDoc*() Note that when uploading a document using either of these functions, you are communicating with the documents API, i.e. writely, and you need authentication for that. So if you have a connection for the spreadsheets API, you cannot use that.

Managing Documents

We have been focusing on the contents of documents. The Google Docs API allows us to manage the collection of documents. It provides functionality to upload and delete/remove documents, rename documents, create folders and move documents into folders. The function *addFolder*() allows us to create a new folder. This is created at the top-level. We can then move it to a different folder. The function *moveToFolder*() does this. *addFolder*() takes one or more names and a connection and creates the folders with these names.

R
```
f = addFolder(c("foo", "bar"), con)
```
The results are *GoogleFolder* objects. We can use these as target/destinations in calls to *moveToFolder*() . For example, we can move the folder bar into foo with

R
```
moveToFolder(f$bar, f$foo)
```

We can change meta data, such as the title, of the document. This is done via the operators *$*<-*() and *[<-*() . (The accessor methods are not supported.) Given a document **doc** we set, e.g., the title with

R
```
doc$title = "new title"
```
We can also use

R
```
doc["title"] = "new title"
```
The benefit of the second approach (apart from allowing variables, e.g. `var = "title"; doc[ var ] = value`) is that we can set multiple values in a single call, e.g.

R
```
doc["title", "author"] = list("new title", c(name = "Bob", email = "bob@bob.com"))
```

Comment

The Google documents and spreadsheets servics are interesting. The Web-based nature has several attractive aspects. However, the interactive tools are currently quite limited relative to regular office applications.

There is little functionality for working richly with styles. The APIs are also reasonably limited. Furthermore, some of the documentation is slightly unclear and even incorrect, e.g. the code for some examples do not correspond to what is being discussed in the text, the batch editing section talks about POST, but in fact PUT works and POST does not appear to. While developing some of the functionality in this package, the Google service claimed it was experiencing technical difficulties and was unable to list all my documents. So I am not ready to trade-in my office tools (not that I use them very much anyway!), but the notion of publishing "live" documents is appealing. The IDynDocs package has a different take on "live".