API v1.5

I took some time to work through the new API with some help and data from @AaronDavidSchneider.

Here's what we found.

Concepts

Storage and files

Metadata, Content, pages, and highlight files/folders still exist. The changes to the API are not that drastic.

Everything is stored in GCS. Specifically, 64 character hashes are used as ids to retrieve GCS paths from the API.

Uploading and downloading files are done through shortlived authenticated GCS URLs that the API returns.

Root File Index

The **root file index** points to the root file that is currently considered the source of truth root file for the remarkable account in the cloud.

It is a one-line file that contains the GCS path that points to the root file:

```
59ee28e34c6637a9d296d6593c7b350d438a747ee39fd99316cdf55d0b4536c4
```

Root File

The root file is stored in GCS and is composed of newline delimited rows.

Each row has columns that are `:` delimited.

The first few lines look like this:
```
```

3
29ee28e34c6637a9d996d6593c7b350d438a747ee39fd99316cdf55d0b4536c4:80000000:315d9db7-ea15-4d9e-b2f9-1ca6e070129b:8:0
e03b13372e18a712a7d7970fa52435fbcee421ff0ecbb85ba5e05cc0c58cb8a8:80000000:078efaa4-1189-4b65-a2d2-c78b269024a9:4:0
...
```

Each entry in the **root file** corresponds to a document's index file.

The format for each line in the root file is as follows:

```
{GCS path identifier for index file}:8000000:{document/folder uuid on device}:{number of entries in index file}:0
```

At the moment, we are not sure what the 8000000 or what the row's last number corresponds to.

Index file

An index file is found by following the GCS path identifier for the document in the root file.

The index file is similar to the root file; however, the format of each row is different.

An example index file for document with doc id `2a7324c7-899d-4b44-ad27-26061b4ec679` looks like:

```
3
d983b8cf7964e02324e66c743e390ffac6f974a13afb824d392ab88c1157b59f:0:2a7324c7-899d-4b44-ad27-26061b4ec679.content:0:951
3a9bfe07bef282b120b542f44c3b96701671e6528557e2f1a5a284b4faa76bc7:0:2a7324c7-899d-4b44-ad27-26061b4ec679.metadata:0:315
081718ceefb66454bc3241cc304a184c05ad7dc938293c84d9db9c5ed9c19ef2:0:2a7324c7-899d-4b44-ad27-26061b4ec679.pagedata:0:72
2c4d7ac2c7fd506c3c0e644c21887685e9b4873b2f2f74e15623ddb97756e67a:0:2a7324c7-899d-4b44-ad27-26061b4ec679.pdf:0:799411
```

Each line in the index file corresponds to a file associated with the document. The format of each line is currently interpreted as:

```
```
```

{GCS identifier}:0:{document/folder id}:0:{size of the file}
```

Note, we still do not know what two `0` columns mean for the entries.

## Data files

All the data files you would expect, such as metadata, content, page data ... etc. Can all be found via the GCS path in the index file.

## A format?

The root and index files feel like database table dumps.

Both root files and index files start with 3 and then a new line. The format seems familiar to me, but I can't quite remember if this is a serialization format for a database/library. If someone knows, please tell us. It could prove helpful.

We still don't know what some columns represent in both the root and index files.

# API

With an understanding of each type of file, let's quickly go over how we interact with the API to access the files.

## Constants

SERVICE_MANAGER_URL =
"https://service-manager-production-dot-remarkable-production.appspot.com"
API_URL = https://rm-blob-storage-prod.appspot.com/api/v1
API_DOWNLOAD = API_URL + "/signed-urls/downloads"
API_UPLOAD = API_URL + "/signed-urls/uploads"
SYNC_COMPLETE = API_URL + "/sync-complete"

## Authentication

Authentication occurs the same way as the old API and at the same endpoint.
```

We send a one-time code in, receive a `device token` and then renew it to get a `user token`.

We can then use the resulting user token in our `Authorization: Bearer {user_token}` header.

API endpoint

To get the API endpoint, we send an authenticated GET request to the `SERVICE_MANAGER_URL` with the following parameters:

```
environment=production
apiVer=1
```

This returns:

```
{
  "Status": "OK",
  "Host": "https://rm-blob-storage-prod.appspot.com"
}
```

The `Host` key indicates the API endpoint for us to use.

Requesting a file

To get a file, we need to ask the API for the GCS path of the file.

We need to make sure we're properly authenticated and then send a post request with the following JSON body to the API_DOWNLOAD endpoint:

```
{
  "http_method": "GET",
  "relative_path": "3aaa107b0ecb398205d5b043f9633c83daf431fac2ffe322fc968ee7e1bbe04f"
}
```

Note that the relative path here is the GCS path for the file you want to get a link to.

In response, the API will send back a JSON blob:

```
{
  "relative_path": "3aaa107b0ecb398205d5b043f9633c83daf431fac2ffe322fc968ee7e1bbe04f",
  "url":
"https://storage.googleapis.com/rm-blob-storage-bucket-prod/.../3aaa107b0ecb398205d5b043f9633c83daf431fac2ffe322fc968ee7e1bbe04f...",
  "expires": "2021-07-28T17:24:43.755364159Z",
  "method": "GET"
}
```

Sending a `GET` request to the URL in the response will return the file's contents.

Root file index

Everything needs to start with the root file --- the root file links to the index files and those to the data files.

To get the root file, we first need to know the GCS path for the file. That path is stored in the root file index.

To get the root file index, you perform a request for the file with the relative path `root`.

Root file

The root file can be found by performing a request for the file with the GCS path listed in the root file index above.

Index file

You guessed it. The index file can be found by performing a request for the file with the GCS path listed in the root file for the document.

Data files

You can get actual data files by querying the GCS path listed in the index files.

Uploading files

Uploading a file is straightforward after understanding how we read files.

To upload a file, we send a POST request with the following JSON blob to the API_UPLOAD endpoint:
```
{
  "http_method": "PUT",
  "relative_path":
"8db46f9d472363e5bbea37e91eb7ae18c73145e1eb8c9396aa3405a1d3c0cdab"
}
```
Note the difference in the HTTP method from reading. Here we ask the API to allow us to PUT a file.

The `relative_path` here is generated by the client.

The API should respond with a URL that we can upload our file to. The response should resemble:
```
{
  "relative_path":
"8db46f9d472363e5bbea37e91eb7ae18c73145e1eb8c9396aa3405a1d3c0cdab",
  "url":
"https://storage.googleapis.com/rm-blob-storage-bucket-prod/.../8db46f9d472363e5bbea37e91e
b7ae18c73145e1eb8c9396aa3405a1d3c0cdab...",
  "expires": "2021-07-28T17:33:47.625664959Z",
  "method": "PUT"
}
```

To upload a file, we send a `PUT` request to the URL returned. The body of that request is the file we would like to upload.

Uploading a document

To upload a document, you will need to upload:
- A PDF
- A content file
- A metadata file
- A page data file

The client should send a `PUT` request for each file in the list.

An index file will need to be made and added to the root file to have the data available in subsequent API requests.

Uploading an index file

Uploading an index file involves grabbing the GCS paths used to upload the data files and constructing an index file that matches the definition of the index files above. Each line should represent one of the files uploaded. The client will need to generate a document id.

Once the index file is constructed, it should also be uploaded as a file to the GCS.

Uploading a root file

Uploading a root file is similar to uploading an index file.

You will need to grab the current root file from the API and add a line for the index file you created.

When requesting the current root file index from the API, make sure to record the value of the `x-goog-generation` header from the root file index GCS response. The generation value is required when uploading the root file index next.

Once the new root file has been created, you can upload it to GCS.

Uploading the root file index

Finally, you must upload a new root file index to indicate to the API that it should use the new root file going forward.

The root file index upload process is slightly different than uploading other files. You will need to send a post request to the API_UPLOAD with the following json blob:

```
{
  "generation": "1627493476159831",
  "http_method": "PUT",
  "relative_path": "root"
}
```

```

The generation is the value of the `x-goog-generation` header when requesting the root file index previously. Presumably, this helps the Remarkable sort out the order of operations on their side.

The request should return a blob similar to other upload requests where you can upload a file with a single line containing the GCS path of the root file you uploaded.

### Sync complete

The client should send an empty POST request to the SYNC_COMPLETE endpoint whenever the root file is changed.

# Modifying

## Removing a document/file

Removing a document involves creating a new root file that does not contain the index file of the document you would like to remove.

To remove a file from a document, upload a new index file that does not contain an entry for the file and update the root file with the new index file created.

## Moving a file

Moving a file is a bit different than what you would expect.

In general, you will be creating a new index file that contains the new metadata file with the proper parent set.

The critical difference is that when uploading the metadata file, the `parent_hash` key should be added to the request, the value should be the GCS path of the old index file for the document.

For example:
- Document X has an index file at `abc` and lives in folder 'A'
- We want to move X to 'B'
```

- We upload the new metadata file, but for the request to get a GCS path to upload to, we do:
```
{
 "http_method": "PUT",
 "parent_hash":
"e9b52e66cb45a4d02fb3eab92cfa99b619ddd735eab73761aa123d55b1aae925",
 "relative_path":
"304221b0d9f17a9ab638deae2ddc9ab1e18388831abcbe7ef846669fd3d6565a"
}
```

Moving a file to the trash

Moving a file to the trash is the same as moving a file; however, the parent field in the metadata should be set to `trash`.