fhircrackr: Download FHIR resources

2021-05-12

This vignette covers the following topics:

- Building FHIR search requests
- Downloading resources from a FHIR server
- Authentication
- Search via POST
- Dealing with HTTP errors
- Saving the downloaded bundles
- Dealing with large data sets
- Downloading the capability statement

Before running any of the following code, you need to load the `fhircrackr` package:

```
library(fhircrackr)
```

FHIR search requests

To download FHIR resources from the server, you need to specify which resources you want with a *FHIR search request*. If you are already familiar with the topic and prefer writing your FHIR search request yourself, you can just define your search request as a simple string that you provide to `fhir_search()`. In that case, however, no checking of spelling mistakes of resource types and URL encoding will be done for you. If you are comfortable with this, you can skip the following paragraph, as the first part of this vignette introduces the basics of FHIR search and some functions to build valid FHIR search requests with `fhircrackr`.

A FHIR search request will mostly have the form `[base]/[type]?parameter(s)`, where `[base]` is the base URL to the FHIR server you are trying to access, `[type]` refers to the type of resource you are looking for and `parameter(s)` characterize specific properties those resources should have. The function `fhir_url()` offers a solution to bring those three components together correctly, taking care of proper formatting for you.

In the simplest case, `fhir_url()` takes only the base url and the resource type you are looking for like this:

```
fhir_url(url = "http://hapi.fhir.org/baseR4",
         resource = "Patient")
# An object of class "fhir_url"
# [1] "http://hapi.fhir.org/baseR4/Patient"
```

Internally, `fhir_resource_type` is called to check the type you provided against list of all currently available resource types can be found at https://hl7.org/FHIR/resourcelist.html. Case errors are corrected automatically and the function throws a warning, if the resource type doesn't match the list under hl7.org:

```
fhir_resource_type("Patient") #Correct
# A fhir_resource_type object: Patient
fhir_resource_type("medicationstatement") #corrected
# A fhir_resource_type object: MedicationStatement
```

Beside telling the server which resource type to give back, the resource type also determines the kinds of search parameters that are allowed. Search parameters are use to further qualify the resources you want to get back, e.g by restricting the search result to Patient resources of female patients only.

You can add zero, one, or multiple search parameters to the request. If you don't give any parameters, the search will just return all resources of the specified type from the server. Search parameters generally come in the form `key = value`. There are a number of resource independent parameters that can be found under https://www.hl7.org/fhir/search.html#Summary. These parameters usually have a `_` at the beginning. `"_sort" = "status"` for examples sorts the results by their status, `"_include" = "Observation:patient"`, will include the linked Patient resources in a search for Observation resources.

Apart from the resource independent parameters, there are also resource dependent parameters referring to elements specific to that resource. These parameters come without a `_` and you can find a list of them at the end of every resource site e.g. at https://www.hl7.org/fhir/patient.html#search for the Patient resource. An example of such a parameter would be `"birthdate" = "lt2000-01-01"` for patients born before the year 2000 or `"gender" = "female"` to get female patients only.

You can add search parameters to you request in a named list or named character vector this:

```
request <- fhir_url(url = "http://hapi.fhir.org/baseR4",
        resource = "Patient",
        parameters = list("birthdate" = "lt2000-01-01",
                          "code" = "http://loinc.org|1751-1")
                    )

request
# An object of class "fhir_url"
# [1] "http://hapi.fhir.org/baseR4/Patient?birthdate=lt2000-01-01&code=http://
# loinc.org%7C1751-1"
```

As you can see,fhir_url()performs automatic url encoding and the|is transformed to%7C‘.

Accessing the current request

Whenever you call `fhir_url()` or `fhir_search()`, the corresponding FHIR search request will be saved implicitly and can be accessed with `fhir_current_request()`

If you call `fhir_search()` without providing an explicit request, the function will automatically call `fhir_current_request()`.

Download FHIR resources from a server

To download resources from a server, you use the function `fhir_search()` and provide a FHIR search request.

Basic request

We will start with a very simple example and use `fhir_search()` to download Patient resources from a public HAPI server:

```
request <- fhir_url(url = "https://hapi.fhir.org/baseR4",
                    resource = "Patient")

patient_bundles <- fhir_search(request=request,
                               max_bundles=2, verbose = 0)
```

In general, a FHIR search request returns a *bundle* of the resources you requested. If there are a lot of resources matching your request, the search result isn't returned in one big bundle but distributed over several of them, sometimes called *pages*, the size of which is determined by the FHIR server. If the argument

`max_bundles` is set to its default `Inf`, `fhir_search()` will return all available bundles/pages, meaning all resources matching your request. If you set it to `2` as in the example above, the download will stop after the first two bundles. Note that in this case, the result *may not contain all* the resources from the server matching your request, but it can be useful to first look at the first couple of search results before you download all of them.

If you want to connect to a FHIR server that uses basic authentication, you can supply the arguments `username` and `password`. If the server uses some bearer token authentication, you can provide the token in the argument `token`. See below for more information on authentication.

Because servers can sometimes be hard to reach, `fhir_search()` will start five attempts to connect to the server before it gives up. With the arguments `max_attempts` and `delay_between_attempts` you can control this number as well the time interval between attempts.

As you can see in the next block of code, `fhir_search()` returns an object of class `fhir_bundle_list` where each element represents one bundle of resources, so a list of two in our case:

```
patient_bundles
# An object of class "fhir_bundle_list"
# [[1]]
# A fhir_bundle_xml object
# No. of entries : 20
# Self Link: http://hapi.fhir.org/baseR4/Patient
# Next Link: http://hapi.fhir.org/baseR4?_getpages=ce958386-53d0-4042-888c-cad53bf5d5a1&_getpagesoffset
#
# {xml_node}
# <Bundle>
#  [1] <id value="ce958386-53d0-4042-888c-cad53bf5d5a1"/>
#  [2] <meta>\n  <lastUpdated value="2021-05-10T12:12:43.317+00:00"/>\n</meta>
#  [3] <type value="searchset"/>
#  [4] <link>\n  <relation value="self"/>\n  <url value="http://hapi.fhir.org/b ...
#  [5] <link>\n  <relation value="next"/>\n  <url value="http://hapi.fhir.org/b ...
#  [6] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837602"/ ...
#  [7] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/example-r ...
#  [8] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837624"/ ...
#  [9] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837626"/ ...
# [10] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837631"/ ...
# [11] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837716"/ ...
# [12] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837720"/ ...
# [13] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837714"/ ...
# [14] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837721"/ ...
# [15] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837722"/ ...
# [16] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837723"/ ...
# [17] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837724"/ ...
# [18] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/cfsb16116 ...
# [19] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837736"/ ...
# [20] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837737"/ ...
# ...
#
# [[2]]
# A fhir_bundle_xml object
# No. of entries : 20
# Self Link: http://hapi.fhir.org/baseR4?_getpages=ce958386-53d0-4042-888c-cad53bf5d5a1&_getpagesoffset
# Next Link: http://hapi.fhir.org/baseR4?_getpages=ce958386-53d0-4042-888c-cad53bf5d5a1&_getpagesoffset
#
# {xml_node}
```

```
# <Bundle>
#  [1] <id value="ce958386-53d0-4042-888c-cad53bf5d5a1"/>
#  [2] <meta>\n  <lastUpdated value="2021-05-10T12:12:43.317+00:00"/>\n</meta>
#  [3] <type value="searchset"/>
#  [4] <link>\n  <relation value="self"/>\n  <url value="http://hapi.fhir.org/b ...
#  [5] <link>\n  <relation value="next"/>\n  <url value="http://hapi.fhir.org/b ...
#  [6] <link>\n  <relation value="previous"/>\n  <url value="http://hapi.fhir.o ...
#  [7] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837760"/ ...
#  [8] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837766"/ ...
#  [9] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837768"/ ...
# [10] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837781"/ ...
# [11] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837783"/ ...
# [12] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837784"/ ...
# [13] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837787"/ ...
# [14] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837788"/ ...
# [15] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837789"/ ...
# [16] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837790"/ ...
# [17] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837791"/ ...
# [18] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837792"/ ...
# [19] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837793"/ ...
# [20] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837794"/ ...
# ...
```

If for some reason you cannot connect to a FHIR server at the moment but want to explore the bundles anyway, the package provides an example list of bundles containing Patient resources. See `?patient_bundles` for how to use it.

More than one resource type

In many cases, you will want to download different types of FHIR resources belonging together. For example you might want to download all MedicationStatement resources with the snomed code `429374003` and also download the Patient resources these MedicationStatements refer to. The FHIR search request to do this can be built like this:

```
request <- fhir_url(url = "https://hapi.fhir.org/baseR4/",
                    resource = "MedicationStatement",
                    parameters = list("code"="http://snomed.info/ct|429374003",
                                      "_include"="MedicationStatement:subject")
)
```

Then you provide the request to `fhir_search()`:

```
medication_bundles <- fhir_search(request = request, max_bundles = 3)
```

These bundles now contain two types of resources, MedicationStatement resources as well as Patient resources. If you want to have a look at the bundles, it is not very useful to print them to the console. Instead just save them as xml-files to a directory of you choice and look at the resources there:

```
fhir_save(medication_bundles, directory = "MyProject/medicationBundles")
```

If you want to have a look at a bundle like this but don't have access to a FHIR server at the moment, check out `?medication_bundles`.

Authentication

If your FHIR server is protected with some kind of bearer token authentication, `fhir_search()` lets you provide the token as a string or as an object of class `Token` from the `httr` package. You can

use `fhir_authenticate()` to create a token generated by an OAuth2/OpenID Connect process. See `?fhir_authenticate` for more information on that topic.

Search via POST

The default behaviour of `fhir_search()` is to send the FHIR search request as a `GET` request to the server. In some special cases, however, it can be useful to use the `POST` based search described here instead. This is mostly the case when the URL of you FHIR search request gets long enough to exceed the allowed url length. A common scenario for this would be a request querying an explicit list of identifiers. Let's for example say you are looking for the following list of patient identifiers:

```
ids <- c("72622884-0a09-4ea9-9a91-685bce3b0fe3",
         "2ca48b68-a641-4be7-a39d-9ffe2691a29a",
         "8bcdd92d-5f96-4e07-9f6a-e22a3591ee30",
         "2067558f-c9ed-489a-9c2f-7387bb3426a2",
         "5077b4b0-07c9-4d03-b9ec-1f9f218f8239")
```

You can use them comma separated in the value of the `identifier` search parameter like this:

```
id_string <- paste(ids, collapse =",")
```

But this string would make the FHIR search request URL very long, especially if it is combined with additional other search parameters.

In a search via POST, the search parameters (everything that would usually follow the resource type after the ?) can be transferred to a body of type `application/x-www-form-urlencoded` and sent via POST. A body of this kind can be created the same way the parameters are usually given to the `parameters` argument of `fhir_url()`, i.e. as a named list or character:

```
body <- fhir_body(content = list("identifier" = id_string,
                                 "_revinclude" = "Observation:patient")
)
```

The body will then automatically be assigned the content type `application/x-www-form-urlencoded`. If you provide a body like this in `fhir_search()`, the url in request should **only** contain the base URL and the resource type. The function will automatically amend it with the suffix `_search` and perform a POST:

```
url <- fhir_url(url = "https://hapi.fhir.org/baseR4/",
                resource = "Patient")

bundles <- fhir_search(request = url, body = body)
```

Dealing with HTTP Errors

`fhir_search()` internally sends a `GET` or `POST` request to the server. If anything goes wrong, e.g. because your request wasn't valid or the server caused an error, the result of you request will be a HTTP error. `fhir_search()` will print the error code along with some suggestions for the most common errors to the console.

To get more detailed information on the error response, you can pass a string with a file name to the argument `log_errors` . This will write a log with error information to the specified file.

```
medication_bundles <- fhir_search(request = request,
                                  max_bundles = 3,
                                  log_errors = "myErrorFile")
```

Saving the downloaded bundles

There are two ways of saving the FHIR bundles you downloaded: Either you save them as R objects, or you write them to an xml file.

Save bundles as R objects

If you want to save the list of downloaded bundles as an `.rda` or `.RData` file, you can't just use R's `save()` or `save_image()` on it, because this will break the external pointers in the xml objects representing your bundles. Instead, you have to serialize the bundles before saving and unserialize them after loading. For single xml objects the package `xml2` provides serialization functions. For convenience, however, `fhircrackr` provides the functions `fhir_serialize()` and `fhir_unserialize()` that can be used directly on the bundles returned by `fhir_search()`:

```
#serialize bundles
serialized_bundles <- fhir_serialize(patient_bundles)

#have a look at them
head(serialized_bundles[[1]])
# [1] 58 0a 00 00 00 03
```

```
#create temporary directory for saving
temp_dir <- tempdir()

#save
save(serialized_bundles, file=paste0(temp_dir, "/bundles.rda"))
```

If you load this bundle again, you have to unserialize it before you can work with it:

```
#load bundles
load(paste0(temp_dir, "/bundles.rda"))
```

```
#unserialize
bundles <- fhir_unserialize(serialized_bundles)

#have a look
bundles
# An object of class "fhir_bundle_list"
# [[1]]
# A fhir_bundle_xml object
# No. of entries : 20
# Self Link: http://hapi.fhir.org/baseR4/Patient
# Next Link: http://hapi.fhir.org/baseR4?_getpages=ce958386-53d0-4042-888c-cad53bf5d5a1&_getpagesoffset
#
# {xml_node}
# <Bundle>
#  [1] <id value="ce958386-53d0-4042-888c-cad53bf5d5a1"/>
#  [2] <meta>\n  <lastUpdated value="2021-05-10T12:12:43.317+00:00"/>\n</meta>
#  [3] <type value="searchset"/>
#  [4] <link>\n  <relation value="self"/>\n  <url value="http://hapi.fhir.org/b ...
#  [5] <link>\n  <relation value="next"/>\n  <url value="http://hapi.fhir.org/b ...
#  [6] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837602"/ ...
#  [7] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/example-r ...
#  [8] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837624"/ ...
#  [9] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837626"/ ...
# [10] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837631"/ ...
# [11] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837716"/ ...
```

```
# [12] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837720"/ ...
# [13] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837714"/ ...
# [14] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837721"/ ...
# [15] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837722"/ ...
# [16] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837723"/ ...
# [17] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837724"/ ...
# [18] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/cfsb16116 ...
# [19] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837736"/ ...
# [20] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837737"/ ...
# ...
#
# [[2]]
# A fhir_bundle_xml object
# No. of entries : 20
# Self Link: http://hapi.fhir.org/baseR4?_getpages=ce958386-53d0-4042-888c-cad53bf5d5a1&_getpagesoffset
# Next Link: http://hapi.fhir.org/baseR4?_getpages=ce958386-53d0-4042-888c-cad53bf5d5a1&_getpagesoffset
#
# {xml_node}
# <Bundle>
#  [1] <id value="ce958386-53d0-4042-888c-cad53bf5d5a1"/>
#  [2] <meta>\n  <lastUpdated value="2021-05-10T12:12:43.317+00:00"/>\n</meta>
#  [3] <type value="searchset"/>
#  [4] <link>\n  <relation value="self"/>\n  <url value="http://hapi.fhir.org/b ...
#  [5] <link>\n  <relation value="next"/>\n  <url value="http://hapi.fhir.org/b ...
#  [6] <link>\n  <relation value="previous"/>\n  <url value="http://hapi.fhir.o ...
#  [7] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837760"/ ...
#  [8] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837766"/ ...
#  [9] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837768"/ ...
# [10] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837781"/ ...
# [11] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837783"/ ...
# [12] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837784"/ ...
# [13] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837787"/ ...
# [14] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837788"/ ...
# [15] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837789"/ ...
# [16] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837790"/ ...
# [17] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837791"/ ...
# [18] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837792"/ ...
# [19] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837793"/ ...
# [20] <entry>\n  <fullUrl value="http://hapi.fhir.org/baseR4/Patient/1837794"/ ...
# ...
```

After unserialization, the pointers are restored and you can continue to work with the bundles. Note that the example bundles `medication_bundles` and `patient_bundles` that are provided with the `fhircrackr` package are also provided in their serialized form and have to be unserialized as described on their help page.

Save and load bundles as xml files

If you want to store the bundles in xml files instead of R objects, you can use the functions `fhir_save()` and `fhir_load()`. `fhir_save()` takes a list of bundles in form of xml objects (as returned by `fhir_search()`) and writes them into the directory specified in the argument `directory`. Each bundle is saved as a separate xml-file. If the folder defined in `directory` doesn't exist, it is created in the current working directory.

```
#save bundles as xml files
fhir_save(patient_bundles, directory=temp_dir)
```

To read bundles saved with `fhir_save()` back into R, you can use `fhir_load()`:

```
bundles <- fhir_load(temp_dir)
```

`fhir_load()` takes the name of the directory (or path to it) as its only argument. All xml-files in this directory will be read into R and returned as a list of bundles in xml format just as returned by `fhir_search()`.

Dealing with large data sets

If you want to download a lot of resources from a server, you might run into several problems.

First of all, downloading a lot of resources will require a lot of time, depending on the performance of your FHIR server. Because `fhir_search()` essentially runs a loop pulling bundle after bundle, downloads can usually be accelerated if the bundle size is increased, because that way we can lower the number of requests to the server. You can achieve this by adding `_count=` parameter to your FHIR search request. `http://hapi.fhir.org/baseR4/Patient?_count=500` for example will pull patient resources in bundles of 500 resources from the server.

A problem that is also related to the number of requests to the server is that sometimes servers might crash when too many requests are sent to them in a row. The third problem is that large amounts of resources can at some point exceed the working memory you have available. There are two solutions to the problem of crashing servers and working memory:

1. Use the save_to_disc argument of fhir_search()

If you pass the name of a directory to the argument `save_to_disc` in your call to `fhir_search()`, the bundles will not be combined in a bundle list that is returned when the downloading is done, but will instead be saved as xml-files to the directory specified in the argument `directory` one by one. If the directory you specified doesn't exist yet, `fhir_search()` will create it for you. This way, the R session will only have to keep one bundle at a time in the working memory and if the server crashes halfway trough, all bundles up to the crash are safely saved in your directory. You can later load them using `fhir_load()`:

```
request <- fhir_url(url = "http://hapi.fhir.org/baseR4",
                    resource = "Patient")

fhir_search(request, max_bundles = 10,
            save_to_disc= "MyProject/downloadedBundles")

bundles<- fhir_load("MyProject/downloadedBundles")
```

2. Use fhir_next_bundle_url()

Alternatively, you can also use `fhir_next_bundle_url()`. This function returns the url to the next bundle from you most recent call to `fhir_search()`:

To get a better overview, we can split this very long link along the `&`:

```
strsplit(fhir_next_bundle_url(), "&")
# [[1]]
# [1] "http://hapi.fhir.org/baseR4?_getpages=0be4d713-a4db-4c27-b384-b772deabcbc4"
# [2] "_getpagesoffset=200"
# [3] "_count=20"
# [4] "_pretty=true"
# [5] "_bundletype=searchset"
```

You can see two interesting numbers: `_count=20` tells you that the queried hapi server has a default bundle size of 20. `getpagesoffset=200` tells you that the bundle referred to in this link starts after resource no. 200, which makes sense since the `fhir_search()` request above downloaded 10 bundles with 20 resources

each, i.e. 200 resources. If you use this link in a new call to `fhir_search`, the download will start from this bundle (i.e. the 11th bundle with resources 201-220) and will go on to the following bundles from there.

When there is no next bundle (because all available resources have been downloaded), `fhir_next_bundle_url()` returns `NULL`.

If a download with `fhir_search()` is interrupted due to a server error somewhere in between, you can use `fhir_next_bundle_url()` to see where the download was interrupted.

You can also use this function to avoid memory issues. Th following block of code utilizes `fhir_next_bundle_url()` to download all available Observation resources in small batches of 10 bundles that are immediately cracked and saved before the next batch of bundles is downloaded. Note that this example can be very time consuming if there are a lot of resources on the server, to limit the number of iterations uncomment the lines of code that have been commented out here:

```
#Starting fhir search request
url <- fhir_url(url = "http://hapi.fhir.org/baseR4",
                resource = "Observation",
                parameters = list("_count"="500"))

#count <- 0

df_description <- fhir_df_description(resource = "Observation")

while(!is.null(url)){

    #load 10 bundles
    bundles <- fhir_search(url, max_bundles = 10)

    #crack bundles
    dfs <- fhir_crack(bundles, df_description)

    #save cracked bundle to RData-file (can be exchanged by other data type)
    save(tables, file = paste0(tempdir(),"/table_", count, ".RData"))

    #retrieve starting point for next 10 bundles
    url <- fhir_next_bundle_url()

    # count <- count + 1
    # if(count >= 20) {break}
}
```

Download Capability Statement

The capability statement documents a set of capabilities (behaviors) of a FHIR Server for a particular version of FHIR. You can download this statement using the function `fhir_capability_statement()`:

```
cap <- fhir_capability_statement("http://hapi.fhir.org/baseR4", verbose = 0)
```

`fhir_capability_statement()` takes the base URL of a FHIR server and returns a list of three data frames containing all information from the capability statement of this server. The first one is called `Meta` and contains some general server information. The second is called `Rest` and contains information on the operations the server implements. The third is called `resources` and gives information on the resource types and associated parameters the server supports. This information can be useful to determine, for example, which FHIR search parameters are implemented in you FHIR server.

Next steps

To learn about how `fhircrackr` allows you to convert the downloaded FHIR resources into data.frames/data.tables, see the vignette on flattening FHIR resources.