Vignette

Gowtham Rao

2020-06-11

Contents


```
library(ROhdsiWebApi)
library(magrittr)

if (Sys.getenv("baseUrl") == '') {
  server <- "ohdsiBaseUrl"
} else {server <- "baseUrl"}
baseUrl <- Sys.getenv(server)
```

ROhdsiWebApi is part of HADES.

1 Introduction

From Package Readme

> ROhdsiWebApi is a R based interface to 'WebApi' (OHDSI RESTful services), and performs GET/PULL/POST/DELETE calls via the WebApi. All objects starting from R or output to R - are analysis ready R-objects like list and data.frame. The package handles the intermediary steps by converting R-objects to JSON and vice versa. To ensure r-objects are analysis ready, the objects are type converted where possible, e.g. date/date time are converted from string to POSIXct.
>
> This package makes reproducible research easier, by offering ability to retrieve detailed study specifications, transport study specifications from one instance to another, programmatically

invoke the generation of a sequence of steps that are part of a study, manage running studies in batch mode.

This document will attempt to explain how ROhdsiWebApi maybe used to achieve reproducible research.

2 WebApi configurations and ROhdsiWebApi

To successfully use ROhdsiWebApi, it is necessary to have an active 'WebApi' endpoint with a known baseUrl such as "http://server.org:80/WebAPI". 'WebApi' has many functional categories.

To ensure reproducibility of work it is best to know the version of the WebApi (i.e. Atlas backend) being used. An easy way to do that is (and output maybe included in your study results)

```
version <- ROhdsiWebApi:::getWebApiVersion(baseUrl = baseUrl)
message1 <- paste0('This Vignette was created using WebApi version: ', version, ' on baseUrl: ', baseUrl
cdmSources <- ROhdsiWebApi::getCdmSources(baseUrl = baseUrl)
priorityVocabulary <- ROhdsiWebApi::getPriorityVocabularyKey(baseUrl = baseUrl)
```

The object `version` will show your webApi version. Example: This Vignette was created using WebApi version: 2.7.4 on baseUrl: http://api.ohdsi.org:80/WebAPI. The CDM had the following source data configured: .

```
cdmSources
#> # A tibble: 3 x 7
#>   sourceId sourceName sourceKey sourceDialect cdmDatabaseSche~ vocabDatabaseSc~
#>      <int> <chr>      <chr>     <chr>         <chr>            <chr>
#> 1        4 Common Ev~ CEM       postgresql    <NA>             unrestricted
#> 2        6 SYNPUF 1K  SYNPUF1K  postgresql    synpuf1k         unrestricted
#> 3        5 SYNPUF 5%  SYNPUF5P~ postgresql    synpuf5pct       unrestricted
#> # ... with 1 more variable: resultsDatabaseSchema <chr>
```

The priority vocabulary for the WebApi is SYNPUF5PCT.

We can also perform checks on the WebApi, example - we may want to see if the 'HCUP' & 'SYNPUF1K' is a valid SourceKey in the current webApi.

```
ROhdsiWebApi::isValidSourceKey(sourceKeys = c('HCUP', 'SYNPUF1K'), baseUrl = baseUrl)
#> [1] FALSE  TRUE
```

2.1 WebApi Analytical categories.

WebApi maybe considered to have certain modular analytic categories. ROhdsiWebApi supports the following categories:

```
#> # A tibble: 7 x 2
#>   `Category`      Features
#>   <chr>           <chr>
#> 1 ConceptSet      Functions for interfacing with ConceptSet in WebApi
#> 2 Cohort          Functions for interfacing with Cohort in WebApi
#> 3 IncidenceRate   Functions for interfacing with IncidenceRate in WebApi
#> 4 Estimation      Functions for interfacing with Estimation in WebApi
#> 5 Prediction      Functions for interfacing with Prediction in WebApi
#> 6 Characterization Functions for interfacing with Characterization in WebApi
#> 7 Pathway         Functions for interfacing with Pathway in WebApi
```

3 Framework of ROhdsiWebApi

ROhdsiWebApi maybe better understood by having atleast a high level understanding of CRUD framework for WebApi, i.e. the GET, PUT, DELETE, POST calls to the API. See the documentation of the WebApi.

For each supported category, ROhdsiWebAPi performs GET, PUT, DELETE, POST calls to WebApi in background. The details of what calls are actually performed is less important to an analyst, but it is useful to understand the naming conventions of ROhdsiWebApi.

3.1 Naming conventions of ROhdsiWebApi

Most functions in ROhdsiWebApi start with an action oriented 'verb' - such as

Most of the functions start with the following verbs:

```
#> # A tibble: 12 x 2
#>    `Function Verb` `Number Of Functions`
#>    <chr>                           <int>
#>  1 Get                                37
#>  2 Isvalid                             9
#>  3 Delete                              8
#>  4 Post                                8
#>  5 Detect                              7
#>  6 Exists                              7
#>  7 Cancel                              5
#>  8 Invoke                              5
#>  9 Insert                              2
#> 10 Convert                             1
#> 11 Create                              1
#> 12 Resolve                             1
```

A function to get `Definition` is `getDefinitionMetaData` function. This is a general function that is able to get the Metadata for all specifications within a category.

```
ROhdsiWebApi::getDefinitionsMetadata(baseUrl = baseUrl,
                                     category = 'cohort') %>%
  dplyr::arrange(.data$id) %>%
  dplyr::rename_all(.funs = SqlRender::camelCaseToTitleCase) %>%
  tail()
#> # A tibble: 6 x 7
#>       Id Name  `Created By` `Created Date`      `Modified By` Description
#>    <int> <chr> <chr>        <dttm>              <chr>         <chr>
#> 1 1.77e6 [YM]~ ""           2015-03-06 20:49:00 ""            <NA>
#> 2 1.77e6 [C2Q~ ""           2015-03-06 20:49:00 ""            <NA>
#> 3 1.77e6 YBY_~ ""           2015-03-06 20:49:00 ""            <NA>
#> 4 1.77e6 simv~ ""           2015-03-06 20:49:00 ""            <NA>
#> 5 1.77e6 40-4~ ""           2015-03-06 20:49:00 ""            <NA>
#> 6 1.77e6 New ~ ""           2015-03-06 20:49:00 ""            <NA>
#> # ... with 1 more variable: `Modified Date` <dttm>
```

The same output may be achieved using

```
ROhdsiWebApi::getCohortDefinitionsMetaData(baseUrl = baseUrl) %>%
  dplyr::arrange(.data$id) %>%
  dplyr::rename_all(.funs = SqlRender::camelCaseToTitleCase) %>%
  tail()
#> # A tibble: 6 x 7
```

```
#>       Id Name  `Created By` `Created Date`      `Modified By` Description
#>    <int> <chr> <chr>        <dttm>              <chr>         <chr>
#> 1 1.77e6 [YM]~ ""           2015-03-06 20:49:00 ""            <NA>
#> 2 1.77e6 [C2Q~ ""           2015-03-06 20:49:00 ""            <NA>
#> 3 1.77e6 YBY_~ ""           2015-03-06 20:49:00 ""            <NA>
#> 4 1.77e6 simv~ ""           2015-03-06 20:49:00 ""            <NA>
#> 5 1.77e6 40-4~ ""           2015-03-06 20:49:00 ""            <NA>
#> 6 1.77e6 New ~ ""           2015-03-06 20:49:00 ""            <NA>
#> # ... with 1 more variable: `Modified Date` <dttm>
```

Similar approach may be used for all categories as follows:

```
ROhdsiWebApi::getDefinitionsMetadata(baseUrl = baseUrl,
                                     category = 'estimation') %>%
  dplyr::arrange(.data$id) %>%
  dplyr::rename_all(.funs = SqlRender::camelCaseToTitleCase) %>%
  tail()
#> # A tibble: 6 x 6
#>      Id Name          `Created Date`      `Modified Date`     Type     Description
#>   <int> <chr>         <dttm>              <dttm>              <chr>    <chr>
#> 1   357 "E12-celecox~ 2020-06-06 01:28:30 2020-06-06 01:42:55 Compara~ <NA>
#> 2   358 "New Populat~ 2020-06-06 04:01:52 NA                  Compara~ Lix
#> 3   359 "ACEi versus~ 2020-06-06 04:27:30 2020-06-06 04:28:16 Compara~ Comparing ACE inhib~
#> 4   360 "YBY_ACEi ve~ 2020-06-07 21:16:10 2020-06-07 21:16:56 Compara~ YBY_Comparing ACE i~
#> 5   361 "COPY OF: YB~ 2020-06-07 21:17:38 2020-06-08 01:19:50 Compara~ YBY_Comparing ACE i~
#> 6   362 "PLE - DOAC"  2020-06-09 09:57:28 2020-06-09 10:08:51 Compara~ <NA>
```

```
ROhdsiWebApi::getEstimationDefinitionsMetaData(baseUrl = baseUrl) %>%
  dplyr::arrange(.data$id) %>%
  dplyr::rename_all(.funs = SqlRender::camelCaseToTitleCase) %>%
  tail()
#> # A tibble: 6 x 6
#>      Id Name          `Created Date`      `Modified Date`     Type     Description
#>   <int> <chr>         <dttm>              <dttm>              <chr>    <chr>
#> 1   357 "E12-celecox~ 2020-06-06 01:28:30 2020-06-06 01:42:55 Compara~ <NA>
#> 2   358 "New Populat~ 2020-06-06 04:01:52 NA                  Compara~ Lix
#> 3   359 "ACEi versus~ 2020-06-06 04:27:30 2020-06-06 04:28:16 Compara~ Comparing ACE inhib~
#> 4   360 "YBY_ACEi ve~ 2020-06-07 21:16:10 2020-06-07 21:16:56 Compara~ YBY_Comparing ACE i~
#> 5   361 "COPY OF: YB~ 2020-06-07 21:17:38 2020-06-08 01:19:50 Compara~ YBY_Comparing ACE i~
#> 6   362 "PLE - DOAC"  2020-06-09 09:57:28 2020-06-09 10:08:51 Compara~ <NA>
```

This is a generic framework that applies to most WebApi categories, and supports different types of CRUD functionalities like deleteConceptSetDefinition() vs deleteDefinition(category = 'conceptSet').

4 Concept Set

Please review 'Concept sets - The Book of OHDSI'

We commonly post concept set expression into WebApi/Atlas, or try get an expression from Atlas/WebApi based on a conceptSetDefinitionId.

Example: lets say we have concept set expression as follows, that is being used for a Rheumatoid Arthritis study.

```
jsonExpression <- '{
  "items": [
    {
      "concept": {
        "CONCEPT_ID": 81097,
        "CONCEPT_NAME": "Feltys syndrome",
        "STANDARD_CONCEPT": "S",
        "STANDARD_CONCEPT_CAPTION": "Standard",
        "INVALID_REASON": "V",
        "INVALID_REASON_CAPTION": "Valid",
        "CONCEPT_CODE": "57160007",
        "DOMAIN_ID": "Condition",
        "VOCABULARY_ID": "SNOMED",
        "CONCEPT_CLASS_ID": "Clinical Finding"
      },
      "isExcluded": true,
      "includeDescendants": false,
      "includeMapped": false
    },
    {
      "concept": {
        "CONCEPT_ID": 80809,
        "CONCEPT_NAME": "Rheumatoid arthritis",
        "STANDARD_CONCEPT": "S",
        "STANDARD_CONCEPT_CAPTION": "Standard",
        "INVALID_REASON": "V",
        "INVALID_REASON_CAPTION": "Valid",
        "CONCEPT_CODE": "69896004",
        "DOMAIN_ID": "Condition",
        "VOCABULARY_ID": "SNOMED",
        "CONCEPT_CLASS_ID": "Clinical Finding"
      },
      "isExcluded": false,
      "includeDescendants": true,
      "includeMapped": false
    },
    {
      "concept": {
        "CONCEPT_ID": 4035611,
        "CONCEPT_NAME": "Seropositive rheumatoid arthritis",
        "STANDARD_CONCEPT": "S",
        "STANDARD_CONCEPT_CAPTION": "Standard",
        "INVALID_REASON": "V",
        "INVALID_REASON_CAPTION": "Valid",
        "CONCEPT_CODE": "239791005",
        "DOMAIN_ID": "Condition",
        "VOCABULARY_ID": "SNOMED",
        "CONCEPT_CLASS_ID": "Clinical Finding"
      },
      "isExcluded": false,
      "includeDescendants": true,
      "includeMapped": false
    }
```

```
  ]
}'
```

Lets call this concept set expression - '[ROhdsiWebApi Vignette] Rheumatoid Arthritis concept set'.

```
#> Successfully deleted conceptSet definition id 1864163. Request status code: Success: (204) No Conten
#> NULL
```

Note: function does not accept JSON. It needs to be converted to R (list) expression

We can post the concept set expression into WebApi as follows:

```
#> Post ConceptSet definition was successful
```

If successful, we will get a return object as follows into R.

```
#> # A tibble: 1 x 6
#>   createdBy modifiedBy createdDate         modifiedDate              id name
#>   <lgl>     <lgl>      <dttm>              <dttm>                 <int> <chr>
#> 1 NA        NA         2020-06-11 14:04:41 2020-06-11 14:04:41 1864164 [ROhdsiWebApi Vigne~
```

The id of the newly posted concept-set definition is 1864164. We can now use this concept-set for many concept set queries eg.,

4.1 if want to print ready expression of the concept set definition

```
conceptSetDefinition = getConceptSetDefinition(conceptSetId = returnFromPostRequest$id,
                                               baseUrl = baseUrl)
conceptTbl <-
  convertConceptSetDefinitionToTable(conceptSetDefinition)
names(conceptTbl) <-
  SqlRender::camelCaseToTitleCase(names(conceptTbl))
conceptTbl
#> # A tibble: 3 x 13
#>   `Is Excluded` `Include Descen~ `Include Mapped` `Concept Id` `Concept Name`
#>   <lgl>         <lgl>            <lgl>                  <int> <chr>
#> 1 TRUE          FALSE            FALSE                  81097 Felty's syndr~
#> 2 FALSE         TRUE             FALSE                  80809 Rheumatoid ar~
#> 3 FALSE         TRUE             FALSE                4035611 Seropositive ~
#> # ... with 8 more variables: `Standard Concept` <chr>, `Standard Concept Caption` <chr>,
#> #   `Invalid Reason` <chr>, `Invalid Reason Caption` <chr>, `Concept Code` <chr>, `Domain
#> #   Id` <chr>, `Vocabulary Id` <chr>, `Concept Class Id` <chr>
```

`createConceptSetWorkbook` maybe used to create an Excel workbook of the concept set.

If we want a list of all conceptId's (including descendants) from the concept set expression

```
resolvedConcepts = resolveConceptSet(conceptSetDefinition = conceptSetDefinition, baseUrl = baseUrl)
print("Note: Showing only the first 10 concept id's")
#> [1] "Note: Showing only the first 10 concept id's"
resolvedConcepts[1:10]
#>  [1]   80809 4035427 4035611 4103516 4114439 4114440 4114441 4114442 4114444 4115050
```

The concept set expression json expression can be recaptured from WebApi as follows

```
json <-
  getConceptSetDefinition(baseUrl = baseUrl,
                          conceptSetId = returnFromPostRequest$id
```

```
                          )$expression %>%
  RJSONIO::toJSON(pretty = TRUE)
```

5 Cohorts/Characterization/Incidence rate

Please review 'What is a cohort - The Book of OHDSI'.

> We define a cohort as a set of persons who satisfy one or more inclusion criteria for a duration of time. The term cohort is often interchanged with the term phenotype. Cohorts are used throughout OHDSI analytical tools and network studies as the primary building blocks for executing a research question.
>
> A cohort is defined as the set of persons satisfying one or more inclusion criteria for a duration of time. One person may qualify for one cohort multiple times during non-overlapping time intervals. Cohorts are constructed in ATLAS by specifying cohort entry criteria and cohort exit criteria. Cohort entry criteria involve selecting one or more initial events, which determine the start date for cohort entry, and optionally specifying additional inclusion criteria which filter to the qualifying events. Cohort exit criteria are applied to each cohort entry record to determine the end date when the person's episode no longer qualifies for the cohort.

Cohorts/Characterization/Incidence Rate are WebApi categories, where WebApi manages the execution of generations.

Example: We may want to know if a certain cohort specification has been generated by checking cohort generation status `getCohortGenerationInformation(baseUrl = baseUrl, cohortId= 4234)`. If a cohort is not previously generated, it may be generated using `invokeCohortSetGeneration(baseUrl = baseUrl, cohortId = 4234, sourceKey = 'HCUP')` . If it is already generated, we can extract its output of cohort generation using `getCohortResults(baseUrl, cohortId = 4234)`.

6 Characterization

Please review 'Characterization - The Book of OHDSI'.

7 Population Level Effect Estimation (TO Do)

Please review 'Population Level Effect Estimation - The Book of OHDSI'.

8 Patient Level Prediction (TO Do)

Please review 'Patient Level Prediction - The Book of OHDSI'.