dseconnect REST-API

In diesem Kapitel finden Sie Beispiele für die Nutzung der dseconnect REST-API in R-Skripten.

Die Dokumentation zur dseconnect REST-API finden sie als API-Spezifikation unter folgendem Link: **DSEconnect Rest-API**

Beispiel: Verwendung der REST API über R-Skripte

library(httr)

```
portal_url <- "http://0.0.0.0:9000/backend/"
portal_login_endpoint <- "conditionmonitoring/user/login.json"
```

Login in Portal

```
login_credentials = list(name = "admin", password = "12345")
# Note: With httr, Cookies are automatically persisted between requests to the same domain
r <- POST(paste0(portal_url, portal_login_endpoint),
          body = login_credentials,
          encode = "json")

status_code(r)

## [1] 200
```

Make request to auth.apitoken.rest endpoint to get a apitoken

In a Shiny App that runs inside YUNA, the api_token would be delivered in the URL as a URLParameter e.g.: `http://myshinyapp.com/?apitoken=12345-ABCD-THIS-IS-A-API-TOKEN`

```
api_token_endpoint = "de.eoda.dse.portal.auth.apitoken.rest/"

api_token_response = POST(paste0(portal_url, api_token_endpoint))

api_token <- content(api_token_response, encoding = "utf-8")

api_token

## [1] "47a6339f-93ec-438c-9118-7d788a1db34f"
```

Use the token to login in the APIToken endpoint

```
login_result <-
  POST(paste0(portal_url, api_token_endpoint, "login"),
       query = list(token = api_token))

status_code(login_result)

## [1] 202
```

Fetch a data query for a given data id, user role and reference tag

```
data_query_endpoint = "de.eoda.dse.portal.dataquery.rest/"

dataid = "qy_InstBasis_Overview"

response_dataid <-
  GET(paste0(portal_url, data_query_endpoint, dataid),
      query = list(role = "System_Admin"))
```

When no refTag is given, it will be used the default reftag. A refTag can be included in the query parameters, e.g.:

```
GET(paste0(portal_url, data_query_endpoint, dataid), query = list(role="System_Admin", refTag= "Demo"))
```

```
content(response_dataid)
```

```
## $id
## [1] 102
##
## $name
## [1] "qy_InstBasis_Overview"
##
## $type
## [1] "StoredProcedure"
##
## $content
## [1] "<QueryBuilder>\n  <exec>\n    <procedure>DSE-DataDB</procedure>\n    <procedure>data</procedure>\n
<procedure>sp_GetTableDataAccordingToLanguageParameter</procedure>\n    <parameters>\n      <parameter>\n
<id>currentLanguage</id>\n        <parameter>languageParameter</parameter>\n        <type>VARCHAR</type>\n      <
/parameter>\n      <parameter>\n        <id>filter2</id>\n        <parameter>InstBasis</parameter>\n
<type>VARCHAR</type>\n      </parameter>\n    </parameters>\n  </exec>\n</QueryBuilder>"
##
## $queryTablePath
## [1] "DSE-DataDB data"
##
## $filter
## [1] TRUE
##
## $refTag
## NULL
##
## $cancelable
## $cancelable$present
## [1] FALSE
##
##
## $metaData
## $metaData$typeDefinitions
## $metaData$typeDefinitions$ProductGroup
## $metaData$typeDefinitions$ProductGroup$type
## [1] "Translatable"
##
##
## $metaData$typeDefinitions$MachineType
## $metaData$typeDefinitions$MachineType$type
## [1] "Translatable"
##
##
##
## $metaData$description
## [1] ""
##
##
## $params
## named list()
##
## $roleId
## [1] 1
##
## $queryId
## [1] 47
```

GET Request to build and execute a query

Getting a data table

```
response_dataid_exec <-
  GET(
    paste0(
      portal_url,
      data_query_endpoint,
      "qy_InstBasis_Overview",
      "/exec"
    ),
    query = list(
      "_role" = "System_Admin",
      "_currentLanguage" = "de_DE",
      "_converter" = "columnconverter"
    ) # Column converter will deliver data as columns,
    # which is more pratical to convert to R dataframes
  )

get_columns_as_data_frame <- function(httr_response) {
  stop_for_status(httr_response)
  data_columns <-
    content(httr_response, as = "parsed", simplifyVector = TRUE)$dataQueryResult$columns
  return(data.frame(data_columns, stringsAsFactors = FALSE))
}

data_as_df <- get_columns_as_data_frame(response_dataid_exec)

str(data_as_df)

## 'data.frame':    20 obs. of  26 variables:
##  $ ProductionStartDate: num  1.34e+12 1.34e+12 1.43e+12 1.43e+12 1.17e+12 ...
##  $ ProductGroup       : chr  "Airplane Engine" "Airplane Engine" "Airplane Engine" "Airplane Engine" ...
##  $ ProductionCounter  : int  9678 1234 542 963 28 45 386 4328 238 578 ...
##  $ LastUpdate         : num  1.48e+12 1.45e+12 1.48e+12 1.48e+12 1.45e+12 ...
##  $ LocationCountry    : chr  "DE" "DE" "DE" "DE" ...
##  $ LastServiceMission : num  1.41e+12 1.41e+12 1.49e+12 1.49e+12 1.25e+12 ...
##  $ ParentEquipmentNo  : chr  "Airhansa 1234" "Airhansa 1234" "Airhansa5678" "Airhansa5678" ...
##  $ ObjectType         : chr  "BB-1107C" "BB-1107C" "BB-1107C" "BB-1107C" ...
##  $ EndCustomer        : chr  "Airhansa AG" "Airhansa AG" "Airhansa AG" "Airhansa AG" ...
##  $ CommissioningDate  : num  1.34e+12 1.34e+12 1.42e+12 1.42e+12 1.17e+12 ...
##  $ ShippingDate       : num  1.33e+12 1.33e+12 1.42e+12 1.42e+12 1.17e+12 ...
##  $ LastCmCollect      : num  1.49e+12 1.46e+12 1.48e+12 1.49e+12 1.46e+12 ...
##  $ ID                 : int  6 7 15 16 17 18 10015 10031 10032 10033 ...
##  $ UpdateInfoSpecial  : int  -1 2 1 -1 2 1 -1 -1 -1 -1 ...
##  $ EquipmentFamily    : chr  "BB-110" "BB-110" "BB-110" "BB-110" ...
##  $ Status             : chr  "re" "re" "re" "re" ...
##  $ EquipmentNo        : chr  "BB1107_1" "BB1107_2" "BB1107_3" "BB1107_4" ...
##  $ CustomerNo_AG      : chr  "wert" "wert" "wert" "wert" ...
##  $ MachineTypeTK      : chr  "data.deviceBasicInfo.machineType.turboprop" "data.deviceBasicInfo.machineType.
turboprop" "data.deviceBasicInfo.machineType.turboprop" "data.deviceBasicInfo.machineType.turboprop" ...
##  $ ProductionEndDate  : num  1.42e+12 1.42e+12 NA NA 1.25e+12 ...
##  $ LastServiceMessage : num  1.36e+12 1.46e+12 1.48e+12 1.36e+12 1.46e+12 ...
##  $ MachineType        : chr  "Turboprop" "Turboprop" "Turboprop" "Turboprop" ...
##  $ ProductGroupTK     : chr  "data.deviceBasicInfo.productGroup.airplaneEngine" "data.deviceBasicInfo.
productGroup.airplaneEngine" "data.deviceBasicInfo.productGroup.airplaneEngine" "data.deviceBasicInfo.
productGroup.airplaneEngine" ...
##  $ Generation         : int  2 2 3 3 1 1 4 4 5 4 ...
##  $ UpdateInfoGeneral  : int  -1 2 1 -1 2 1 -1 -1 -1 -1 ...
##  $ Location           : chr  "Berlin" "Berlin" "Berlin" "Berlin" ...
```

Getting just equipment list using a predefined dataid and the filter to get only the ones that have sensor data in our test db

```
response_dataid_equi_list_exec <-
  GET(
    paste0(
      portal_url,
      data_query_endpoint,
      "qy_InstBasis_EquiList",
      "/exec"
    ),
    query = list(
      "_role" = "System_Admin",
      filter2 = "af9c4ff549003fe39c1b3aad38ca4c21f37c9815f0437bec711d06cd7988d58b",
      "_currentLanguage" = "de_DE",
      "_converter" = "columnconverter"
    ) # Column converter will deliver data as columns,
    # which is more pratical to convert to R dataframes
  )


status_code(response_dataid_equi_list_exec)

## [1] 200

equipment_list <-
  get_columns_as_data_frame(response_dataid_equi_list_exec)

equipment_list

##   EquipmentNo
## 1    HA1107_1
## 2    BMK823_1

for (equipment in equipment_list$EquipmentNo) {
  response_dataid_sensor_data_exec <-
    GET(
      paste0(
        portal_url,
        data_query_endpoint,
        "qy_DataFromEquipments",
        "/exec"
      ),
      query = list(
        "_role" = "System_Admin",
        filter2 = "af9c4ff549003fe39c1b3aad38ca4c21f37c9815f0437bec711d06cd7988d58b",
        "_currentLanguage" = "de_DE",
        "_converter" = "columnconverter",
        sensor = "Rd_CMD_Conductivity_Max_R.EC.01.MAX",
        axis = "0",
        element = equipment,
        index = "100",
        operatinghours = "date"
      )
    )
  stop_for_status(response_dataid_sensor_data_exec)
  sensor_data_df <-
    get_columns_as_data_frame(response_dataid_sensor_data_exec)
  sensor_data_df$Timestamp <-
    as.POSIXct(sensor_data_df$xData / 1000,
               tz = "UTC",
               origin = "1970-01-01")

  plot(Value ~ Timestamp, data = sensor_data_df, main = equipment)
}
```