Autodesk® Fusion Manage Reporting using Power BI

A simple guide to visualizing your Fusion Manage data

Contents

Introduction

In today's data-driven world, having a comprehensive view of your information is crucial for making informed decisions. Using a product lifecycle management solution like Autodesk Fusion Manage helps with this by centralizing product development data and processes. This allows for different roles and responsibilities to collaborate with greater accessibility, Fusion Manage includes a lot of PLM workspaces for managing the product lifecycle. Reporting within a single workspace is easy, but what if you need to obtain a holistic view of data across different workspaces or to easily compare records? This guide will show you how you can expose data from Fusion Manage in Power BI.

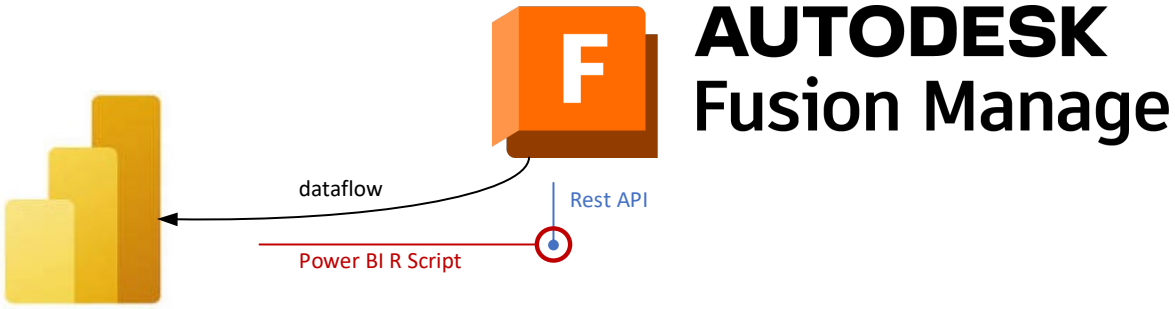
Architectural Overview

Fundamentally, two systems are integrated using the default networking capabilities available in each tool:

- Fusion Manage exposes a generic REST API that can be used to query data.

- Power BI can import data from REST API using an R script.[1]

This can be visualized as follows:



Getting ready to write R code

Although coding can be accomplished via a simple text editor. It is impossible to write code immediately with the correct syntax. An iterative process of writing and testing is typically used to perfect the code. To test the code, we will need to install something which can run the code outside of Power BI.

R is an interpreted language: an interpreter is required to run R code. This interpreter can be obtained via the CRAN[2].

The code to make REST calls and interpret the results requires additional packages. These packages can be installed ahead of time. You can install the modules using the R-Console.

[1] https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-r-scripts
[2] https://cran.r-project.org/

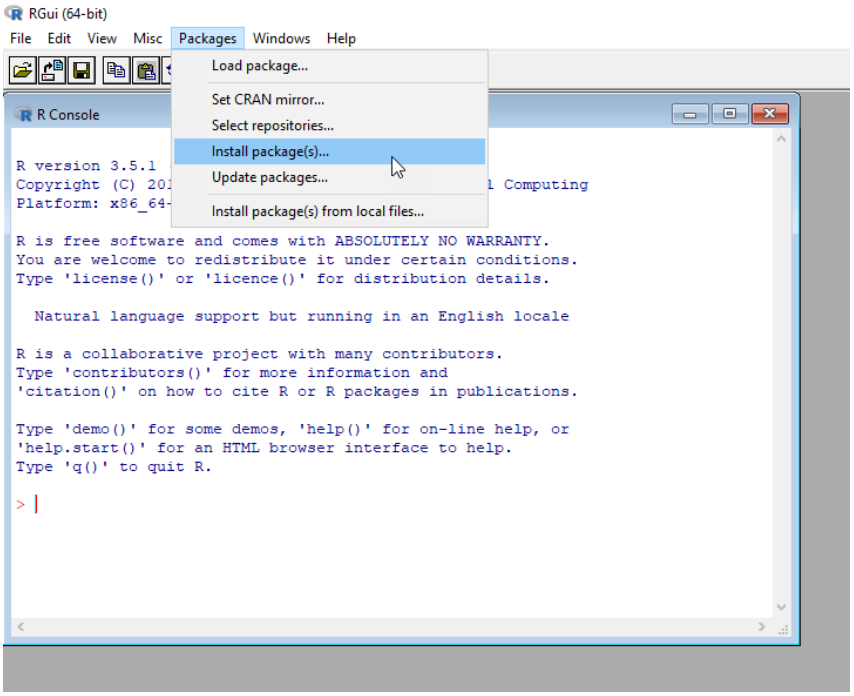


Figure 1: Installing Modules

The packages used in this example are the following.

- **httr** to make the REST API calls from R.
- **jsonlite** to handle the JSON that is returned.

To simplify the process, it is recommended that tools are used to highlight errors, suggest correct syntax, and debug code. These tools are called IDEs (Integrated Development Environments). For R, the recommended IDE is RStudio.[3]

[3] https://posit.co/download/rstudio-desktop/

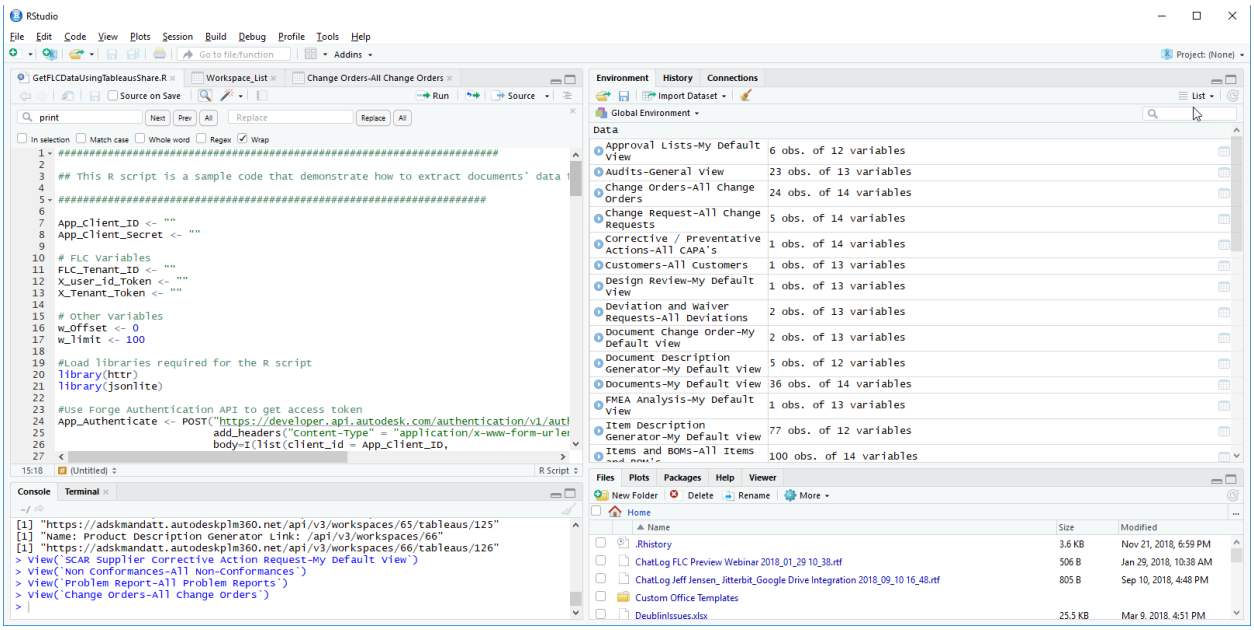


Figure 2: RStudio

R Script

The following code pulls all Fusion Manage data into Power BI.

```
################################################################################

################################################################################
# Load libraries required for the R script
library(httr)
library(jsonlite)

clientid <- ""
clientsecret <- ""
userid <- ""
tenant <- ""

flcURL <- paste ("https://", tenant, ".autodeskplm360.net", sep="")

# concatenate clientid and clientsecret strings using : separator
APP_Base64_String = paste(clientid, clientsecret, sep = ":")

# base64encode the string
Basic_App_Token <- base64_enc(APP_Base64_String)

# FLC Variables
FLC_Tenant_ID <- flcURL
```

```
X_user_id_Token <- userid
X_Tenant_Token <- tenant

# Other Variables
w_Offset <- 0
w_limit <- 999

App_Authenticate <-
  POST(
    'https://developer.api.autodesk.com/authentication/v2/token',
    add_headers(
      'Content-Type' = 'application/x-www-form-urlencoded',
      Authorization = paste("Basic", Basic_App_Token),
      Accept = 'application/json'
    ),
    body = I(
      list("grant_type" = "client_credentials",
           "scope" = "data:read")
    ),
    encode = "form"
  )
Access_Token <-
  paste("Bearer", content(App_Authenticate)$access_token, sep = " ")

print(paste("Basic", Basic_App_Token))


# a list of workspaces.
Get_Workspaces_URL <-
  paste(FLC_Tenant_ID,
        "/api/v3/workspaces?offset=",
        w_Offset,
        "&limit=",
        w_limit,
        "&",
        sep = "")

# print (Get_Workspaces_URL)

Get_Workspaces_Request <- GET(Get_Workspaces_URL,
                              add_headers(
                                .headers = c(
                                  "X-user-id" = X_user_id_Token,
                                  "X-Tenant" = X_Tenant_Token,
                                  "Authorization" = Access_Token
```

```
                              )
                            ))
Get_Workspaces_Data <-
  jsonlite::fromJSON(content(Get_Workspaces_Request, "text", "application/json", "UTF-8"))
Workspace_List <- flatten(data.frame(Get_Workspaces_Data))

# print(Workspace_List)

Parse_Workspace <-
  function(workspace_name,
           workspace_link,
           flag_latest,
           i_offset,
           i_limit) {
    Get_Tableaus_URL <- paste(FLC_Tenant_ID,
                              workspace_link,
                              "/tableaus",
                              sep = "")

    Get_Tableaus_Request <- GET(Get_Tableaus_URL,
                                add_headers(
                                  .headers = c(
                                    "X-user-id" = X_user_id_Token,
                                    "X-Tenant" = X_Tenant_Token,
                                    "Authorization" = Access_Token
                                  )
                                ))
    Get_Tab_Data <-
      jsonlite::fromJSON(content(Get_Tableaus_Request, "text", "application/json", "UTF-8"))
    Tab_List <- flatten(data.frame(Get_Tab_Data))

    tablink <- Tab_List[1, "tableaus.link"]
    tabname <- Tab_List[1, "tableaus.title"]

    Get_Items_URL <- paste(FLC_Tenant_ID,
                           tablink,
                           "?page=1&size=1200",
                           sep = "")

    print(Get_Items_URL)

    tryCatch({
      Get_Items_Request <- GET(Get_Items_URL,
                               add_headers(
                                 .headers = c(
```

```
                                    "X-user-id" = X_user_id_Token,
                                    "X-Tenant" = X_Tenant_Token,
                                    "Authorization" = Access_Token
                                  )
                                ))

    stop_for_status(Get_Items_Request)
  },
  http_error = function(e) {
    ## log error or otherwise recover
    print(e)
  })

  Get_Items_Data <-
    jsonlite::fromJSON(content(Get_Items_Request, "text", "application/json", "UTF-8"))

  X <- flatten((data.frame(Get_Items_Data)))

  Z <- data.frame(
    Date = as.Date(character()),
    File = character(),
    User = character(),
    stringsAsFactors = FALSE
  )

  for (row in 1:nrow(X)) {
    # print(row)

    rsname <- paste(wsname, tabname, "attributes", sep = "-")
    Y <- do.call(rbind.data.frame, X[row, "items.fields"])

    keeps <- c("id", "value")
    Y <- Y[keeps]

    tY <- setNames(data.frame(t(Y[,-1])), Y[, 1])
    Z <- rbind(Z, tY)
  }

  X <- cbind(X, Z)

  # Rename
  assign((paste(wsname, tabname, sep = "-")),
         X,
         envir = .GlobalEnv)
}
```

```
for (row in 1:nrow(Workspace_List)) {
  wslink <- Workspace_List[row, "items.link"]
  wsname <- Workspace_List[row, "items.title"]

  tryCatch({
    print(paste("Name:", wsname,
                "Link:", wslink))

    Parse_Workspace(wsname, wslink, "true", 0, 10)
  },
  error = function(e) {
    cat("An error occurred:", conditionMessage(e), "\n")
    return(NA)  # Return NA in case of an error
  })
}

# Clear Variables
rm(
  clientid,
  clientsecret,
  userid,
  tenant,
  flcURL,
  APP_Base64_String,
  Basic_App_Token,
  FLC_Tenant_ID,
  X_user_id_Token,
  X_Tenant_Token,
  App_Authenticate,
  Access_Token,
  Get_Workspaces_URL,
  Get_Workspaces_Request,
  Get_Workspaces_Data
)
```
[4]

The code starts by setting some variables and then executes as follows.

- Log in using Autodesk Platform Services[5]
- Get a list of workspaces
- For each workspace, get the default view (this is usually in the first element in the tableau array)

[4] Special thanks to Josh Wilson updating the code to use APS v2 Authentication.
[5] https://aps.autodesk.com/

- Get all the data in that view.
- Report on errors (such as no data in that view)

Nuances in the code

Authentication

Authentication is handled with APS. This process requires a few steps.

- Creating an application on the APS website
- Allowing the application to call REST API in the tenant by adding the client ID to the allowed list in the Tenant settings.

Since this is system-to-system communication, a 2-Legged Authentication is used. Consulting the Fusion Manage help pages for more details.[6]

API changes

Overtime, the API will change. The code generically obtains all attributes in a specific workspace view, The order of data in the view can change. This may affect the Power BI reports created with this data.

Paging limits

If the "pagesize" URL parameter is not stipulated in the tableau request, only the first 50 records are retrieved. The code has been updated to retrieve the first 999 records. This is based on a variable that can be increased.

Dates

When dates are imported in Power BI, they appear as subordinate objects in the table. These dates will need further processing to report fully.

Generating Code

Tools like PostMan can generate R code for the API calls.[7] However, it will not provide the code for manipulating the data.

Adding the R Script to Power BI

Connecting Power BI to Fusion Manage is accomplished by pasting the updated R script into Power BI desktop and then publishing the report.

[6] https://help.autodesk.com/view/PLM/ENU/?guid=FLC_RestAPI_v3_API_2_legged_Tutorial_html
[7] https://www.postman.com/

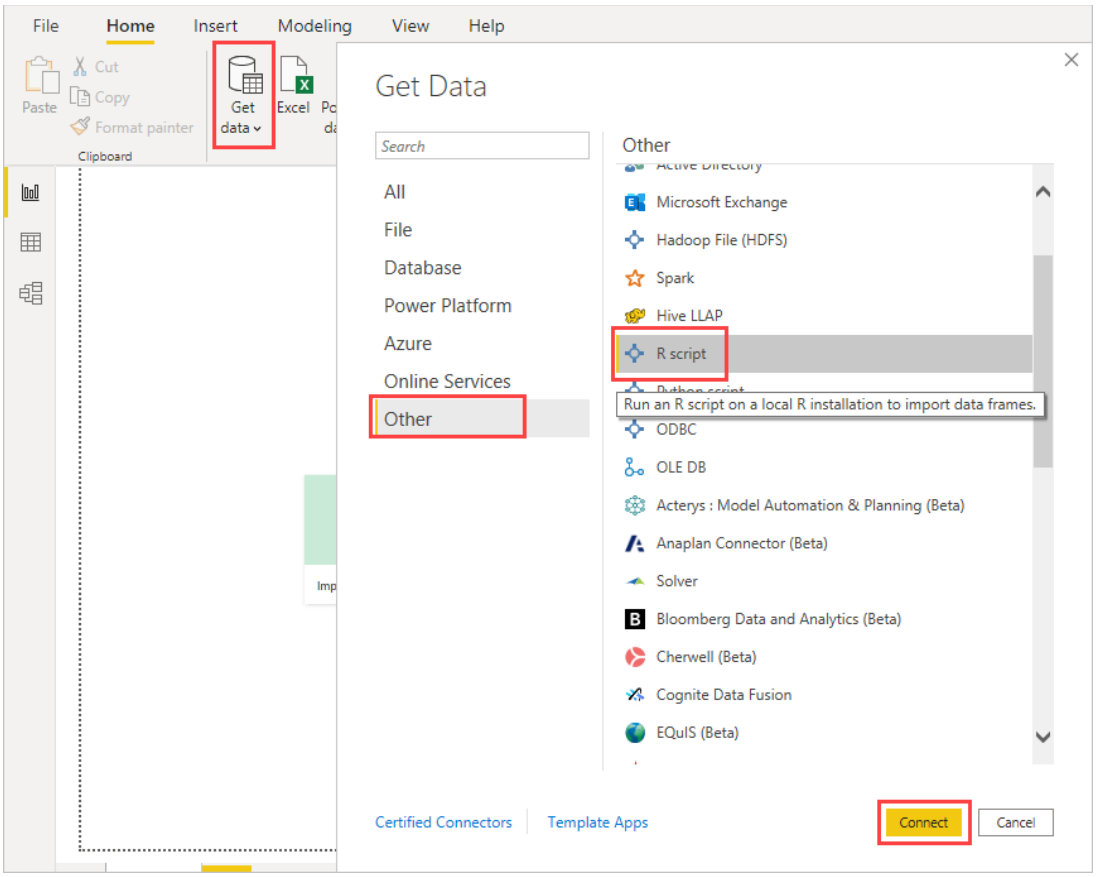


Figure 3: Getting data using R Script

Once imported, the tables appear as follows:

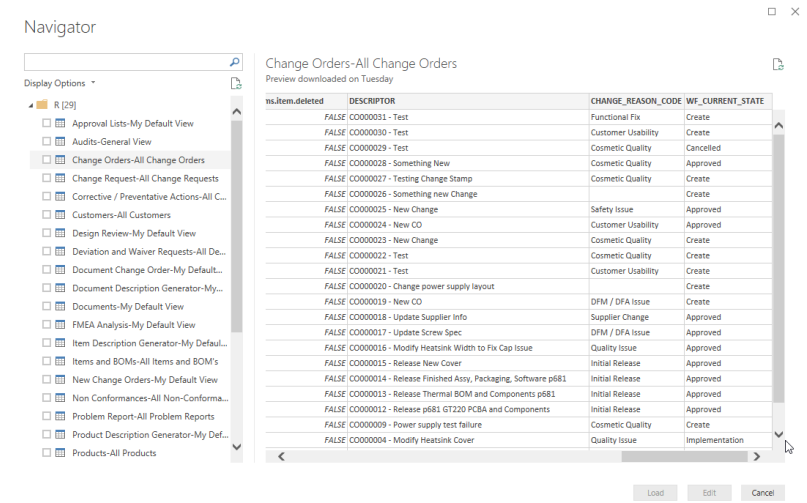


Figure 4: Importing tables from Fusion Manage

AUTODESK

Select each of the tables that you would like to import into Power BI.

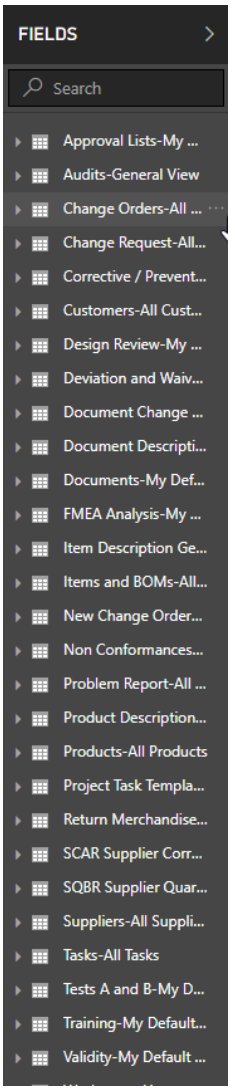


Figure 5: Imported Data

Conclusion

In this guide, we showed how you can expose data from Fusion Manage in Power BI. Power BI was chosen because it is a cloud-based tool that is readily available to Microsoft 365 users. The use of Power BI necessitated the use of R scripting for gathering data. This method can be used with any reporting tool. Although the language used in each of the tools may differ, they should all support pulling data with REST API.