DFS TD - Analytical Lab

Specific technical documentation for the Analytical Lab tests

Summary

Talend Documentation

The following document was created at the beginning of the project and needs to be update but the core idea is still valide:

Silica Analytical Talend

Raw data extraction

Raw data files for all labs are extracted from the servers presented in the following spreadsheet: Asset Tag Spreadsheet

The table below presents all the Analytical tests that are currently included in the Datalake, the source of data and the raw data files format for each test:

Test	Data Sources	File format
CPS	Raw data files	csv
Granulometrie	Raw data files	xls
~~ICP-MS~~	~~Raw data files~~	csv
~~ICP-OS~~	~~Raw data files~~	xls
Porohg	Raw data files	xls
Sedigraph	Raw data files	xls
Mesure Carbone	Raw data files	pdf
~~Infra-Rouge~~	~~Raw data files~~	~~out of scope~~

Talend jobs

Data is processed using Talend and Python. Talend jobs regarding Application Lab can be found in the Talend project **Rnl_Silica_Analytical**

Job **F000_Orch_Flow_Synthesis** is the orchestrator and will call successively all the other jobs a determined order.

Python scripts

A copy of the Python scripts for downloading, parsing and computing new fields and columns for the Application Lab raw data and ELN data can be found here:

However it's recommended to get the last version of the scripts available in the production path: D:\DATA\PROD\RnI\Silica\Permanent. For the moment, **git is not in place for this project.**

Analytical Tests files

CPS

For each raw data file, the Python script **parse_CPS_analysis.py** extracts or calculates the following columns and generates a .csv file:

Column name	Formula
sample_id	extracted from the raw data file
date_measure	extracted from the raw data file. It's the date in which the file was created/generated
diameter_microns	extracted from the raw data file
frequency	extracted from the raw data file
frequency_normalized	We first compute: • max_frequency = Maximum of the column [frequency] Then: [frequency_normalized] = [frequency] * 100 / max_frequency
area	For each row we compute: • min_frequency =The smallest value among [frequency][idx] and [frequency][idx+1] • max_frequency = The greatest value among [frequency][idx] and [frequency][idx+1] Then: area =((([diameter_microns][idx] - [diameter_microns][idx+1]) * min_frequency) + (([diameter_microns][idx] - [diameter_microns][idx+1]) * (max_frequency - min_frequency))) / 2
aggregate	aggregate = [area][idx] + [aggregate][idx-1]
aggregate_normalized	We first compute: • max_aggregate = Maximum value of the column [aggregate] Then: aggregate_normalized = 100 - [aggregate][idx] * 100 / max_aggregate

For each sample, the Python script **compute_CPS_analysis.py** computes the following values (more details can be found in the python script comments):

Value	Formula
sample_id	extracted from the raw data file
date_measure	extracted from the raw data file. It's the date in which the file was created/generated
d10	Inferred using linear regression, by taking the nearest smaller than 10 values of the [aggregate_normalized] column
d10	Inferred using linear regression, by taking the nearest smaller than 16 values of the [aggregate_normalized] column
d50	Inferred using linear regression, by taking the nearest smaller than 50 values of the [aggregate_normalized] column
d84	Inferred using linear regression, by taking the nearest smaller than 84 values of the [aggregate_normalized] column
d90	Inferred using linear regression, by taking the nearest smaller than 90 values of the [aggregate_normalized] column
ld	(d84 - d16) / d50

mode	We first compute: • diameter_at_max_frequency = The value of the column [diameter] corresponding to the maximum of the column [frequency] Then: mode = 1000 * diameter_at_max_frequency

Granulométrie

For granulométrie, two types of raw data files are included in the datalake:

Kinetics

Kinetics files are google spreadsheets that are listed in the google spreadsheet “Dispersibilité 190W”, in the sheet called “Samples”:

https://docs.google.com/spreadsheets/d/1kiPlW5-_a-Eo8fTyP9LmnOsKEZt3gkjw4fYk1TRpHM4/edit#gid=4663305

Sample IDs are taken from the column “Liste des échantillons analysés” and the unique key corresponding to the sample is taken from the column “URL”.

Data for each sample is extracted from the corresponding google spreadsheet and, once the data was extracted, the corresponding cell of the column H of the spreadsheet “Dispersibilité 190W” is updated with the value “done”. If an error occurred for one of the samples and we need to reprocess the corresponding file, we need to suppress the string “done” in column H and rerun the code.

For each sample, the python script **parse_granulo_cinetique.py** extracts the following columns and creates à .csv file:

- **sample_id**
- record_nb
- time
- energy_eff_g_silica_J_g
- Dx_10
- Dx_16
- Dx_50
- Dx_84
- Dx_90
- D_4_3
- D_3_2
- mode
- results_below_1
- results_below_40
- results_below_50
- results_below_75
- mode_0
- mode_1
- mode_2
- mode_count
- mode_percentage_0
- mode_percentage_1
- mode_percentage_2
- obscuration_laser_percentage
- obscuration_bleue_percentage

Differential and Cumulative Curves

For each sample, the python script **parse_granulo_curbes.py** extracts the following columns and creates three different tables:

First Table: Differential curves – from the ***first sheet*** of the excel file we extract the following columns:

- no_enregistrement (4_avg, 1, 2, 3)
- classes_de_taille
- densite_volume

Second Table: Cumulative curves – from the ***second sheet*** of the excel file we extract the following columns:

- no_enregistrement (4_avg, 1, 2, 3)
- classes_de_taille
- densite_volume

Third Table: Table of results – from the ***third sheet*** of the excel file, we extract the following columns: **Note!!!** The Table of results has the same structure as the table of **Kinetics**. Therefore, these two tables are merged in Big Query.

- **record_nb**
- date_heure
- Dx_10

- Dx_16
- Dx_50
- Dx_84
- Dx_90
- D_4_3
- D_3_2
- mode
- results_below_1
- results_below_40
- results_below_50
- results_below_75
- mode_0
- mode_1
- mode_2
- mode_count
- mode_percentage_0
- mode_percentage_1
- mode_percentage_2
- obscuration_laser_percentage
- obscuration_bleue_percentage

ICP (processes not in production as at 22/09/2022)

For both ICP MS and ICP OES, along with raw data files, we need to extract data from the google spreadsheet called “Registre Appareil - ANA-RIC-Paris Balance/micro-onde N°0003531” (feuille 1):

https://docs.google.com/spreadsheets/d/1JRPpjUQdxHKyer7RfvotVq1vTmW1zCD-mykWoqQAND4/edit#gid=0

This will allow us to extract, for each **sample_id**, the fields **volume_liquide** and **masse_echantillon**. These values will be used for computing the concentration of each element from each sample.

ICP MS

For each raw data file, the python script **parse_ICP_masse.py** creates two .csv tables: ICP_MS_details and ICP_MS_summary.

The table ***ICP_MS_details*** contains the following fields:

Column	Formula
sample_id	extracted from the raw data file name
FD	extracted from the raw data file
concentration_1	extracted from the raw data file
concentration_2_read	extracted from the raw data file
masse_echantillon	
volume_liquide	
concentration_2_computed	([concentration_1] * [FD] * volume_liquide * masse_echantillon) /1000
low_limit	0.3 * the value of the first etalon
high_limit	1.65 * the value of the last etalon
warning	For each element: • “warning” if the value of “concentration_1” is higher than or equal to high_limit +/- high_limit * 0.1; • “warning+++” if the value of “concentration_1” is higher than or equal to high_limit; • “warning+++” if the value of “concentration_1” is lower than or equal to low_limit.

The table ***ICP_MS_summary*** contains, for each sample, the value of “concentration_2_computed” for each element of the periodic table of elements (i.e., Al, Ti, Ca, Fe, Mg, Na, S, K, Zn, Sn, Li, P, As, Cd, Ce, Co, Cr, Cu, Hg, Mn, Ni, Pb, Sb, Se, W, Zr, Mo, La, Ba, Bi, V, Be, B, Si, Sc, Ga, Ge, Rb, Sr, Y, Nb, Ru, Rh, Pd, Ag, In, Te, Cs, Hf, Ta, Re, Os, Ir, Pt, Au, Tl, Pr, Nd, Pm, Sm, Eu, Gd, Tb, Dy, Ho, Er, Tm, Yb, Lu).

ICP OES

For each raw data file, the Python script **parse_ICP_optique.py** creates two .csv tables: ICP_OES_details and ICP_OES_summary, by following the same rules as for ICP MS:

The table ***ICP_OES_details*** contains the same fields as the table ICP_MS_details, with one exception: In the ICP_OES_details table, we have a supplementary column called “percentage”. This column is filled only for two elements (i.e., Y and Co), whose values are expressed as a percentage.

The table ***ICP_OES_summary*** contains the same fields as the table ICP_MS_summary.

InfraRouge (removed from project scope)

For each sample, the Python script **parse_InfraRouge.py** creates a .csv file containing the following columns extracted from the raw data file:

- wave_length
- absorbance

Mesure Carbone (not being used as at 22/09/22)

For each sample, the Python script **parse_Mesure_Carbone.py** creates a .csv file containing the following columns extracted from the raw data file:

- carbon_concentration
- sulfur_concentration

Porohg

For each sample, the python script **parse_porohg_analysis.py** creates a .csv file containing the following columns extracted from the raw data file:

- sample_id
- diameter
- cumulative_pore_volume
- smoothed_pore_volume
- cumulative_pore_area

For each sample, the Python script **compute_porohg_analysis.py** creates two tables:

The **First Table** contains the columns extracted from the raw data files and the following computed column:

- The **step** column is computed by applying the following rules:
    - ***“first intrusion”*** – from the beginning of the file and as long as the diameter values are decreasing;
    - ***“extrusion”*** – from the end of the first intrusion step and as long as the diameter values are increasing;
    - ***“second intrusion”*** – from the end of the extrusion step and as long as the diameter values are decreasing
    - ***“end”*** – from the end of the second intrusion and until the end of the file

The **Second Table** contains the following computed values:

Column	**Formula**
sample_id	extracted from the raw data file
date_measure	extracted from the raw data file. It's the date in which the file was created/generated
total_volume	Value of the column [cumulative_pore_volume] at the end of the first intrusion step
v1_micron	We first: • Infer the value of cumulative_pore_volume_m by using linear regression with the two points of the column [diameter] that are closest to 1000 (just before and just after) during the first intrusion. Then: v1_micron = total_volume - cumulative_pore_volume_m
v1_nano	We first: • Infer the value of cumulative_pore_volume_n by using linear regression with the two points of the column [diameter] that are closest to 100 (just before and just after) during the first intrusion. Then: v1_nano = total_volume - cumulative_pore_volume_n

total_volume2	Value of the column (cumulative_pore_volume) at the end of the second intrusion step
vol_agg_micron	We infer the value of vol_agg_micron by using linear regression with the two points of “diameter” that are closest to 1000 (just before and just after) during the second intrusion.
vol_agg_nano	We infer the value of vol_agg_micron by using linear regression with the two points of “diameter” that are closest to 100 (just before and just after) during the second intrusion.
v2_micro	total_volume2 - vol_agg_micro
v2_nano	total_volume2 - vol_agg_nano
mode1	Maximum value of the column [smoothed_pore_volume] where the values of the column [diameter] are <= 100 during the first intrusion
mode1	Maximum value of the column [smoothed_pore_volume] where the values of the column [diameter] are <= 100 during the second intrusion
A1	
B1	
ldp1	(B1 – A1) / mode1
A2	
B2	
ldp2	(B2 – A2) / mode2

Sedigraph

For each sample, the Python script **parse_sedigraph_analysis.py** creates a .csv file containing the following columns extracted or computed from the raw data file:

Column	Formula
sample_id	extracted from the raw data file
date_measure	extracted from the raw data file. It's the date in which the file was created/generated
diameter	extracted from the raw data file
mass_percent	extracted from the raw data file
differential	[mass_percent][idx] - [mass_percent][idx+1]

For each sample, the Python script **compute_sedigraph_analysis.py** creates one table containing the following computed values:

- **d1_micro The value of the column [mass_percent] for which the value of the column [diameter] is closest to 1.**
- **d05_micro The value of the column [mass_percent] for which the value of the column [diameter] is closest to 0.5.**
- **d03_micro The value of the column [mass_percent] for which the value of the column [diameter] is closest to 0.3.**