

# DFS TD - Analytical Lab

Specific technical documentation for the Analytical Lab tests

## Summary

- [Summary](#)
- [Talend Documentation](#)
- [Raw data extraction](#)
- [Talend jobs](#)
- [Python scripts](#)
- [Analytical Tests files](#)
  - [CPS](#)
  - [Granulométrie](#)
  - [ICP \(processes not in production as at 22/09/2022\)](#)
  - [InfraRouge \(removed from project scope\)](#)
  - [Mesure Carbone \(not being used as at 22/09/22\)](#)
  - [Porohg](#)
  - [Sedigraph](#)

## 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
<del>ICP-MS</del>	<del>Raw data files</del>	csv
<del>ICP-OS</del>	<del>Raw data files</del>	xls
Porohg	Raw data files	xls
Sedigraph	Raw data files	xls
Mesure Carbone	Raw data files	pdf
<del>Infra-Rouge</del>	<del>Raw data files</del>	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\Rn\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
<b>sample_id</b>	extracted from the raw data file
<b>date_measure</b>	extracted from the raw data file. It's the date in which the file was created/generated
<b>diameter_microns</b>	extracted from the raw data file
<b>frequency</b>	extracted from the raw data file
<b>frequency_normalized</b>	<p>We first compute:</p> <ul style="list-style-type: none"> <li>max_frequency = Maximum of the column [frequency]</li> </ul> <p>Then:</p> $[\text{frequency\_normalized}] = [\text{frequency}] * 100 / \text{max\_frequency}$
<b>area</b>	<p>For each row we compute:</p> <ul style="list-style-type: none"> <li>min_frequency = The smallest value among [frequency][idx] and [frequency][idx+1]</li> <li>max_frequency = The greatest value among [frequency][idx] and [frequency][idx+1]</li> </ul> <p>Then:</p> $\text{area} = ((([\text{diameter\_microns}][\text{idx}] - [\text{diameter\_microns}][\text{idx}+1]) * \text{min\_frequency}) + (([\text{diameter\_microns}][\text{idx}] - [\text{diameter\_microns}][\text{idx}+1]) * (\text{max\_frequency} - \text{min\_frequency}))) / 2$
<b>aggregate</b>	$\text{aggregate} = [\text{area}][\text{idx}] + [\text{aggregate}][\text{idx}-1]$
<b>aggregate_normalized</b>	<p>We first compute:</p> <ul style="list-style-type: none"> <li>max_aggregate = Maximum value of the column [aggregate]</li> </ul> <p>Then:</p> $\text{aggregate\_normalized} = 100 - [\text{aggregate}][\text{idx}] * 100 / \text{max\_aggregate}$

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

<b>mode</b>	<p>We first compute:</p> <ul style="list-style-type: none"> <li>diameter_at_max_frequency = The value of the column [diameter] corresponding to the maximum of the column [frequency]</li> </ul> <p>Then:</p> <p>mode = 1000 * diameter_at_max_frequency</p>
-------------	--

## 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/1kiPIW5-\\_a-Eo8fTyP9LmnOsKEZt3gkjw4fYk1TRpHM4/edit#gid=4663305](https://docs.google.com/spreadsheets/d/1kiPIW5-_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\_cinétique.py** extracts the following columns and creates a .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
<b>sample_id</b>	extracted from the raw data file name
<b>FD</b>	extracted from the raw data file
<b>concentration_1</b>	extracted from the raw data file
<b>concentration_2_read</b>	extracted from the raw data file
<b>masse_echantillon</b>	
<b>volume_liquide</b>	
<b>concentration_2_computed</b>	$([concentration_1] * [FD] * volume_liquide * masse_echantillon) / 1000$
<b>low_limit</b>	0.3 * the value of the first etalon
<b>high_limit</b>	1.65 * the value of the last etalon
<b>warning</b>	For each element: <ul style="list-style-type: none"> <li>• "warning" if the value of "concentration_1" is higher than or equal to high_limit +/- high_limit * 0.1;</li> <li>• "warning+++" if the value of "concentration_1" is higher than or equal to high_limit;</li> <li>• "warning+++" if the value of "concentration_1" is lower than or equal to low_limit.</li> </ul>

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
<b>sample_id</b>	extracted from the raw data file
<b>date_measured</b>	extracted from the raw data file. It's the date in which the file was created/generated
<b>total_volume</b>	Value of the column [cumulative_pore_volume] at the end of the first intrusion step
<b>v1_micron</b>	<p>We first:</p> <ul style="list-style-type: none"> <li>• 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.</li> </ul> <p>Then:</p> $v1\_micron = total\_volume - cumulative\_pore\_volume\_m$
<b>v1_nano</b>	<p>We first:</p> <ul style="list-style-type: none"> <li>• 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.</li> </ul> <p>Then:</p> $v1\_nano = total\_volume - cumulative\_pore\_volume\_n$

<b>total_volume2</b>	Value of the column (cumulative_pore_volume) at the end of the second intrusion step
<b>vol_agg_micron</b>	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.
<b>vol_agg_nano</b>	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.
<b>v2_micro</b>	total_volume2 - vol_agg_micro
<b>v2_nano</b>	total_volume2 - vol_agg_nano
<b>mode1</b>	Maximum value of the column [smoothed_pore_volume] where the values of the column [diameter] are <= 100 during the first intrusion
<b>mode2</b>	Maximum value of the column [smoothed_pore_volume] where the values of the column [diameter] are <= 100 during the second intrusion
<b>A1</b>	
<b>B1</b>	
<b>ldp1</b>	$(B1 - A1) / mode1$
<b>A2</b>	
<b>B2</b>	
<b>ldp2</b>	$(B2 - A2) / mode2$

## 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
<b>sample_id</b>	extracted from the raw data file
<b>date_measure</b>	extracted from the raw data file. It's the date in which the file was created/generated
<b>diameter</b>	extracted from the raw data file
<b>mass_percent</b>	extracted from the raw data file
<b>differential</b>	$[mass\_percent][idx] - [mass\_percent][idx+1]$

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.