**Deliverable report for**

**Grant Agreement Number 952921**

**Due date of deliverable: 28/02/2021**

**Actual submission date: 08/03/2021**

DELIVERABLE D4.1

PEDR

**Lead beneficiary for this deliverable:**

|  |  |  |
| --- | --- | --- |
| **Dissemination Level:** | | |
| PU | Public |  |
| PP | Restricted to other programme participants (including the Commission Services) |  |
| RE | Restricted to a group specified by the consortium (including the Commission Services) |  |
| CO | Confidential, only for members of the consortium (including the Commission Services) | X |

**This table is for internal use only – to be deleted before submission:**

|  |  |
| --- | --- |
| Version: 2 | Date: 08.03.2021 |
| Draft of the WP Leader | 26.03.2 |
| Commented version for amendment |  |
| Version accepted by the Steering Board |  |
| Report uploaded via Research Participant Portal (QUEST) | dd.mm.yyyy |

**Table of Contents**

[2.1 Structure of the CHADA file (.cha) 5](#_Toc66113681)

[2.2 Structure of the CHADA group file (.chag) 6](#_Toc66113682)

[2.3 CHADA calibration files (.chacal) 6](#_Toc66113683)

[2.4 CHADA model files (.chamod) 7](#_Toc66113684)

[2.5 Listing of CHADA file metadata 9](#_Toc66113685)

[2.6 Steps for initialization of a .cha class instance 9](#_Toc66113686)

[2.7 Steps for initialization of a .chag class instance 9](#_Toc66113687)

[2.8 List of essential files included in a CHADA (.cha) file archive 10](#_Toc66113688)

[2.9 List of attributes of the CHADA group (.chag) class 11](#_Toc66113689)

[**Appendix A: Python pseudocode for CHADA data transformer for background subtraction (chada.base)** 15](#_Toc66113690)

[**Appendix B: Glossary** 17](#_Toc66113691)

*Terms set in italics are defined in the glossary (Appendix B).*

1. **Requirements for the CHADA file format set as part of the CHARISMA proposal**

**CHARISMA proposal Objective 4: *Harmonize Raman characterization data (Ref. WP: 4)***

***Generate an enhanced CHADA (CHAracterisation DAta) structure for Raman Spectroscopy that provides the information to make Raman files and chemometric analyses from different systems (spectrometers and samples) compatible through an algorithm*** or a sequence of ***algorithms.***

The proposal states that the file format should render “*Raman data meaningful and comparable, reflecting only chemical and physical material properties*”. Specifically, the CHADA file format and ecosystem must fulfill the following requirements:

1. Data must be converted to a universal data structure enabling universal comparability.
2. CHADA must be compatible with a future FAIR Raman data repository (WP5).
3. Must enable characterization in real industrial cases (WP6).
4. Each CHADA file must contain a minimal set of metadata that will be defined and structured appropriately.
5. CHADA must avoid information loss (preserve the *Native Data*)
6. The spectral resolution, i.e. the number of *Raman Shift* channels, must be chosen appropriately to realize resolutions of approx. 1 cm-1.
7. An appropriate spectral range must be covered for all current laser sources and spectrometers.
8. CHADA must contain event logs that document all processing steps performed on the *Raw Data*, together with their parameters
9. CHADA data must be readable and processable on large timescales and thus independent of software versions and changes.
10. **Elements of the CHADA ecosystem**

**The proposed harmonized Raman data ecosystem that fulfills above requirements consists of four file types (cf. Figure 1 and Figure 2):**

1. **CHADA file archives (.cha)**
2. **CHADA groups (.chag)**
3. **CHADA calibration files (.chacal)**
4. **CHADA models (.chamod)**

**All *Methods* applicable to CHADA are implemented and stored separately from the data. They can be invoked e.g. by local installation of the CHARISMA library or software *API*, or via a web application running on the CHARISMA server.** The file format description and software protocols can be made accessible to manufacturers of Raman setups for industrial implementation, under an appropriate license offered by the CHARISMA consortium (e.g. a FAIR license).

**CHADA files with interoperable Raman data will be structured as file archives as described below. Individual files will be stored as serialized files in some widely accepted and accessible file format, such as *HDF5* or *JSON*. Since this document describes an initial approach, the specific file format will be defined at a later point.**

## 2.1 Structure of the CHADA file (.cha)

CHADA files are stored as file archives containing several individual files. The archive is data-only and stores information in the form of basic, long-lived data structures, such as *Python* *Lists* and *Dictionaries*. On the other hand, Python classes that implement complex methods, e.g. for data analysis and processing, are provided separate from the data, either through a web server or as a local installation. The latter are subject to updates and changes, that may be necessary e.g. when *Python* functions and formats are updated, or at a point where even the programming language itself may become obsolete.

CHADA file archives contain the Raman data file in the original *Native Format*, together with metadata and a list of applied transformations. They are meant to be permanently stored, shared, and distributed. CHADA files are the basis for CHADA groups and future databases and can be added to the latter. CHADA file archives consist of (cf. Figure 1):

* metadata (static and dynamic blocks; Table 1)
* a copy of the *Native Data* file (typically binary or text)
* a list of data transformers

As data, a CHADA file can hold a single spectrum (0 dimensions plus *Raman Shifts*), a line profile or time series (1D), a Raman map or line profile series (2D), or even volumes or map series (3D). Data being part of a single CHADA file must be recorded with the same instrument in the same mode, and part of the same experiment, i.e. recorded within a certain time period typically no longer than 24h. Since the data originate from the same *Native File*, these prerequisites are generally fulfilled.

The type of data contained in the CHADA files as *Native Data* will depend on structure and content of the *Native Data* file that is typically saved using the OEM software. While it is intended to always store the *Raw Data* in the CHADA file where possible, it may not be available in some cases, so that e.g. *Primary Data* may have to be stored, that has been subject to mathematical modifications.

## 2.2 Structure of the CHADA group file (.chag)

CHADA group files are stored as serialized Python class instances in binary format, using e.g. the Python library *Pickle*. CHADA groups are temporary and established for a specific purpose, e.g. comparing a set of spectra, find common components (decomposition) or the training of a prediction model, such as for the content of a specific nanomarker. Therefore, the longevity of files is not as critical as for CHADA files, so that they may be stored in version-dependent serializations that may become obsolete after an extended period of time.

The set of constituting CHADA (.cha) files is chosen upon group initialization, and the current data – i.e. the (*k, y*) as resulting from the current processing state defined by the transformers list - is converted to a common *Raman Shifts* vector and a data matrix (both numpy.array; cf. Figure 1). On this array, a set of *Methods* can be applied, which generate results such as plots, images, lists, or CHADA models for prediction. A *.chag* file can also be handed targets for model generation. CHADA groups are not meant to be universally comparable, or for permanent storage.

## 2.3 CHADA calibration files (.chacal)

These files contain data transformers acting on the spectrum data (both Raman Shifts k and counts y), such as:

* Spectrum channel-dependent *Raman Shift* calibration –> *k* axis

• Spectrum channel-dependent gain (amplitude) calibration -> *y* axis

• Optical point spread function deconvolution (PSF correction) -> *y* axis

• Spectrometer CCD modulation transfer function deconvolution (MTF correction) -> *y* axis

The files are specific to a certain instrument in a certain mode, and generated by CHADA *Methods* (e.g. chada.k\_calibration()). CHADA files containing data acquired by this instrument and mode can be calibrated using the corresponding .chacal file, and thereby become interoperable and comparable (Figure 2).

## 2.4 CHADA model files (.chamod)

Model files contain a fitted / trained model (e.g. from Scikit Learn) that can e.g. classify a material using its CHADA file, or to predict some material property of the sample, such as nanomarker content. The model file also contains a transformer (*Raman Shifts* range, normalization, …) that is applied to each CHADA file previous to prediction.

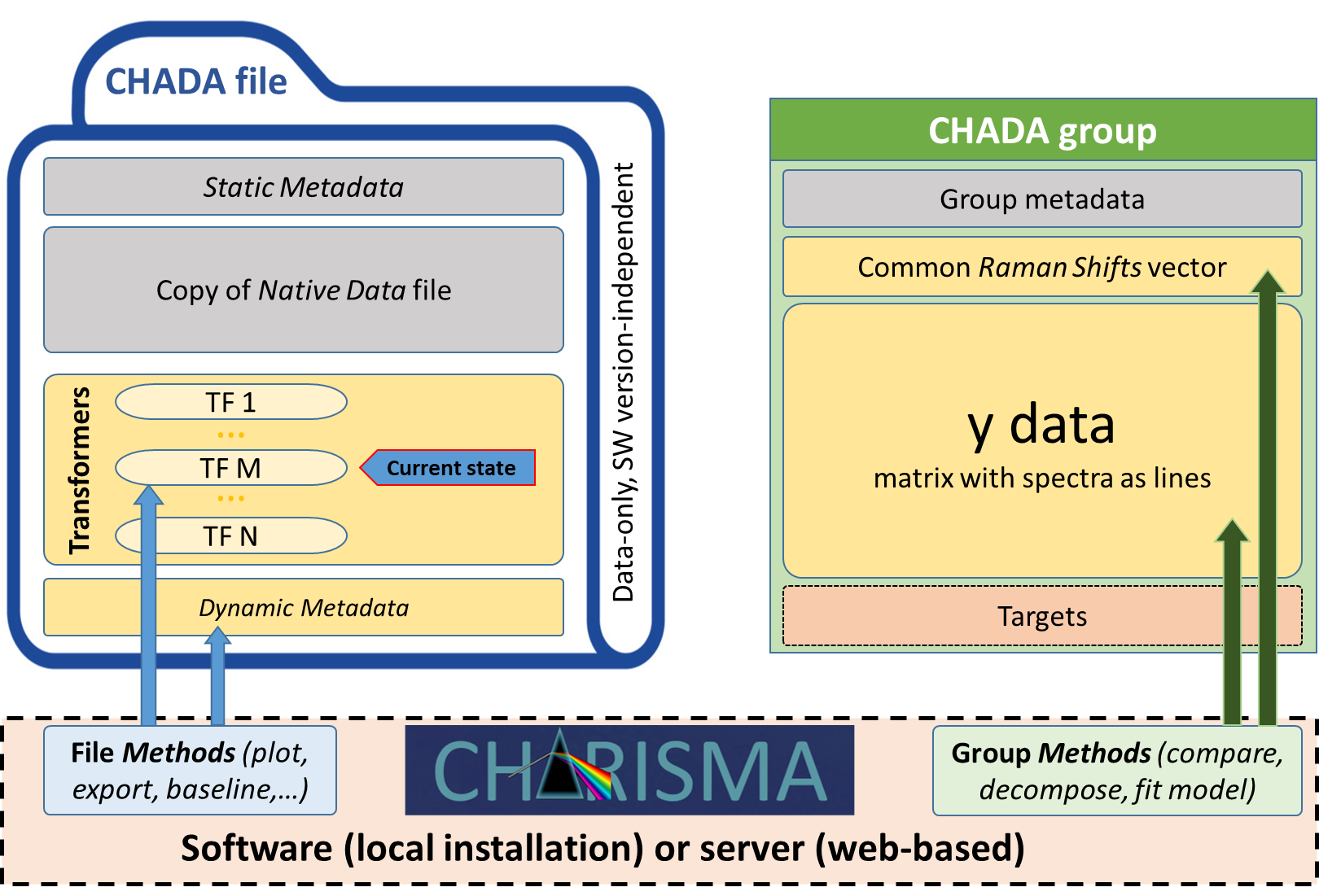


Figure 1: Schematic structure of CHADA (left) and CHADA group files (right). Static elements (unchanged) are gray, while dynamic ones (change) are yellow.

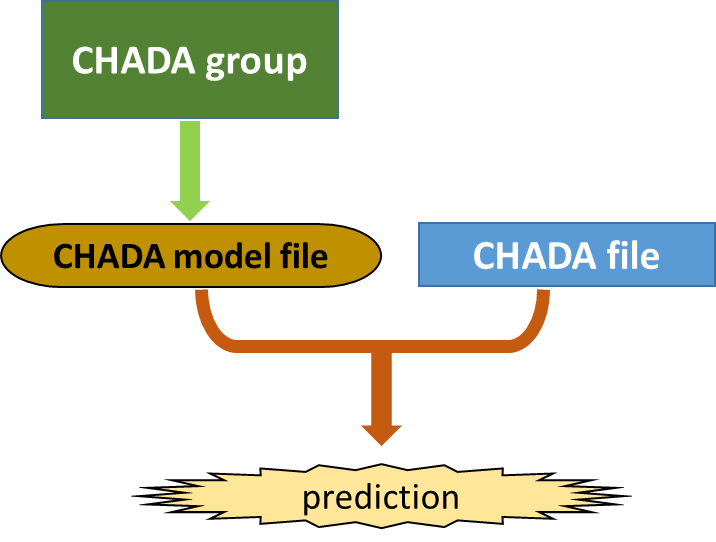
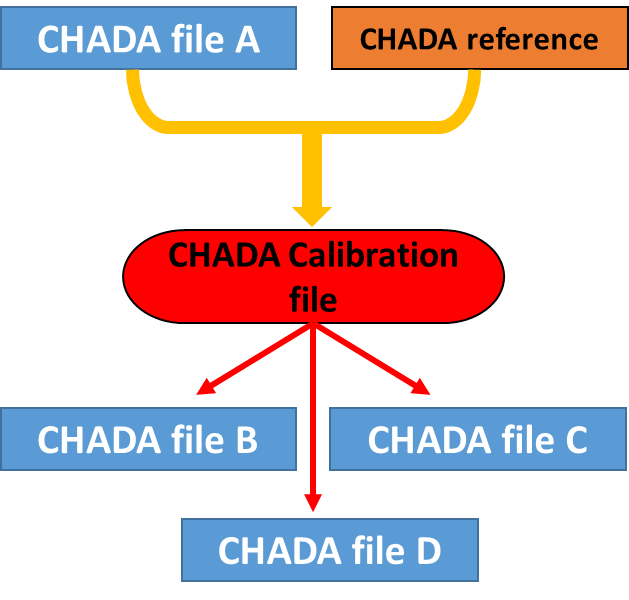
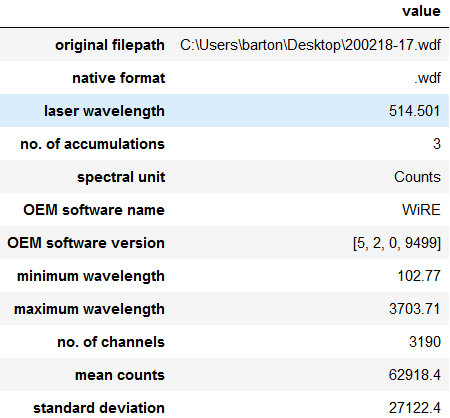
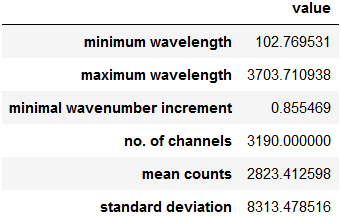
****

Figure 2: Flow diagrams illustrating the generation and use of CHADA calibrations (left) and CHADA models (right). (left) A Raman Shifts calibration for a specific instrument and measuring mode is generated from a CHADA file and a reference measurement of the same calibration sample (alternatively, a list of Raman band positions or intensities). Once generated, a CHADA calibration can be batch-applied to all CHADA data measured with this instrument and mode (files B-D). The same scheme holds for gain, PSF, and MTF calibrations. (right) A model, such as linear regression or neural network, is trained using a CHADA group. Then, the model can be applied to CHADA files in order to predict some material property of the sample, such as nanomarker content, or to classify materials.

## 2.5 Listing of CHADA file metadata

Table 1: Examples for static (left) and dynamic metadata (right) included in a CHADA instance. See also Appendix B.

## 2.6 Steps for initialization of a .cha class instance

1. Create CHADA file archive and include a copy of the *Native Data* file
2. Choose matching native file format reader according to filename extension (.spc, .wdf, .txt, .csv, …), or user specification.
3. Import *Native* file using the matching reader included in the CHARISMA software.
4. Extract metadata from native metadata and spectrum data, store in metadata dictionary, and include in CHADA archive.
5. Check if list of initial transformations has been given by user (e.g. “-b –s –c[310,1890]“ = baseline + smooth + crop Raman Shifts to 310 – 1,890 1/cm).
6. Evoke *Methods* corresponding to initial transformer list, generate dynamic metadata, update transformer list, and store in CHADA archive

## 2.7 Steps for initialization of a .chag class instance

1. Read specified set of CHADA files
2. Store metadata of group members in *chag.metadatas*
3. Add dimensionality of group members (0D, 1D, 2D, …) together with physical dimensions (map x/y in µm, time, …) in *chag.metadatas*
4. From the *Dynamic Metadata* of the file set, find intersection of *Raman Shifts* vectors (only these channels are populated in all spectra, and thus have comparable data) -> set *k\_min* and *k\_max* for group Raman Shifts vector
5. Find minimal increment in *k* vectors of all files -> set increment for group Raman Shifts vector
6. Generate group *Raman Shifts* vector
7. Read all spectra from CHADA member files sequentially, interpolate to group *Raman Shifts* vector, and insert into *chag.y\_data* matrix as lines

## 2.8 List of essential files included in a CHADA (.cha) file archive

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| *'bands',* | Pandas DataFrame or dictionary | Positions, amplitudes etc. for most prominent Raman bands found by the peaks() method. The features measured for Raman bands (position, center of gravity, integral) will be according to existing ASTM standards. |
| *'metadata\_static',* | dictionary | *Static metadata* extracted from *Native Data* file and derived from the data upon CHADA generation |
| *'metadata\_dyn',* | dictionary | *Dynamic Metadata* that will change upon processing |
| *'processing\_state'* | dictionary | State of processing, index to some element of the transformers list |
| *'readers',* | dictionary | Readers corresponding to *Native File* extension |
| *'transformers'* | dictionary | Dictionary containing transformation types and parameters |
| *'background\_model'* | numpy array | background model (y) generated by the chada.base() method |
| *'description'* | string | Description of sample and / or experiment added by the user, or imported from an external file |

## 2.9 List of attributes of the CHADA group (.chag) class

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| *'metadatas'* | list of dictionarys | metadata form the individual CHADA files included in the group |
| *'k\_data'* | numpy array | *k* axis being the intersection of all group member axes, and the minimal increment occuring in all individual axes is used for sampling. |
| *'filenames',* | string | Name of original file |
| *'y\_data'* | numpy array | All *y* data (count) vectors of the spectra in the group. Multi-dimensional data (series, maps) are serialized as lines of the *y\_data* matrix. Data shaped in this manner can be used for a variety of machine learning models included in Python's *ScikitLearn* library. The original dimensionality (pixels and physical) are included in the 'metadatas' attribute. |
| *'description'* | string | Description of sample and / or experiment added by the user, or imported from an external file |
| *'targets'* | list | Optional known properties of the samples, such as composition or ageing state. May serve as targets for CHADA model training. |

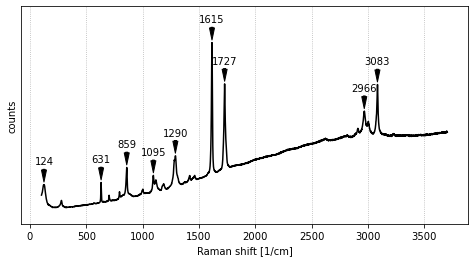
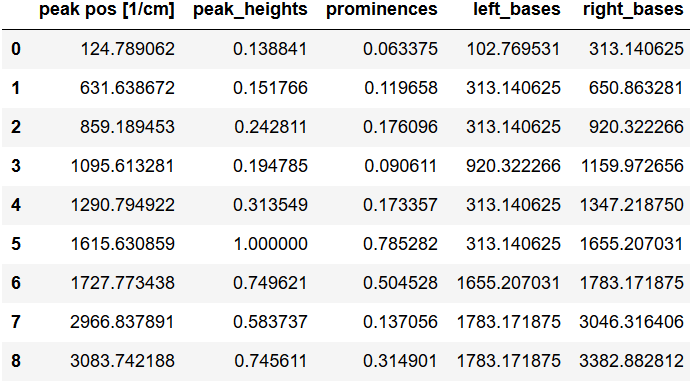
**** 

Figure 3: Demonstrator of the chada.peaks() method. Prominent bands are located and a table of feature statistics is generated as Pandas DataFrame. The latter can easily be evaluated and exported e.g. to Excel or CSV formats.

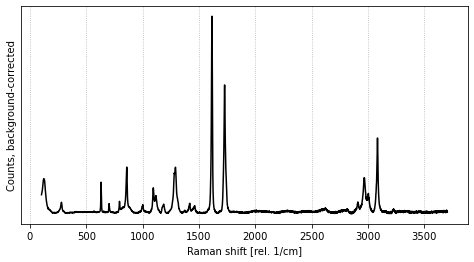
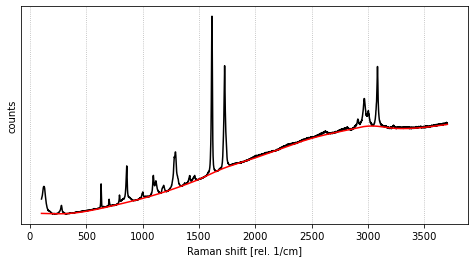


Figure 4: Demonstrator of the chada.base() method. A baseline model is fitted (left) and subtracted from the spectrum (right).

1. **List of use cases for CHADA**

**The following list is not complete and includes only use cases for single spectra (no maps or time series). Methods can be provided for both CHADA and CHADA group files as needed according to the use cases. The *Methods* are provided separately from the CHADA files and groups, e.g. through a web server or local installation of the CHARISMA software or *Python* library.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case** | |  | **CHADA method(s)** |
|  | **SINGLE CHADA FILE** | | |
|  | Single spectrum CHADA files contain native data, meta data, and transformers | | |
| Create CHADA instance | |  | *chada.create()* |
| View / explore spectrum | |  | *chada.plot()* |
| Process spectrum (smooth, background, cosmic ray removal) | |  | *chada.smooth(), chada.base(), chada.crays()* |
| Revert data to the last processing state. *chada.rewind(0)* will revert to the original, unprocessed state. | |  | *chada.rewind()* |
| Save plot as PNG, JPG, PDF,… | |  | *chada.saveplot()* |
| Export spectrum as spc, xls, csv | |  | *chada.export()* |
| Export file processing log as xls, csv, … | |  | *chada.export\_log()* |
| Find bands, extract band statistics | |  | *chada.peaks()* |
| Search spectrum in database | |  | *chada.db\_find()* |
| **Dump original data to disk in *Native Format*** | |  | ***chada.native\_dump()*** |
|  | |  |  |
|  | **CHADA GROUP** | | |
|  | CHADA groups interpolate CHADA spectra to a common *Raman Shift* axis , apply normalization and store data as numpy arrays or pandas DataFrame | | |
| Create CHADA group instance | |  | *chag.\_\_init\_\_()* |
| Compare group of spectra | |  | *chag.compare()* |
| Decompose group of spectra into component spectra | |  | *chag.decompose()* |
| Get group statistics (occurrence of bands, variations) | |  | *chag.stats()* |
| Extract group features (uni- or multivariate) | |  | *chag.features()* |
| Fit linear regression model to known nanomarker content (target) using characteristic band intensity (univariate feature). This will produce a CHADA model (.chamod) instance. | |  | *chag.features(), chag.LR\_model.fit()* |
| Round Robin test comparing intensities and positions of characteristic Raman bands arising from a specific material, using measurements from different sites, instruments, and modes | |  | *chag.normalize(), chag.features(), chag.stats()* |

1. **Properties of CHADA**

* **The CHADA classes and methods are implemented in *Python*, a no-cost programming language that is widely used and accepted in the scientific, data science and industrial R&D sector.**
* **The *Native Data* is always preserved as part of the CHADA (.cha) file archive (requirement #5). It can be reproduced (dumped to disk) in its original format at any point using an appropriate *Method* (*chada.native\_dump()*).**
* **The requirement of sufficient *Raman Shift* range and sampling is always fulfilled, since CHADA works with transformers on the native data, rather than interpolating on a common Raman Shifts axis (requirements #6 & 7). An interpolation (data manipulation) is only performed when a CHADA group is created. In this case, data is interpolated on a *Raman Shifts* axis being the intersection of all group member axes, and the minimal increment occurring in all individual axes is used for sampling.**
* **No data is duplicated upon spectrum data processing. The size of single CHADA file archives always stays comparable to the size of the *Native Data* file, while including the latter.**
* **All data processing steps are reversible, and logged in a user-readable format together with their complete set of parameters. Log files can be generated and exported as Excel table etc.**
* **Complex, machine learning tasks such as decomposition or prediction model training can be implemented and constantly updated for the CHADA group (.chag) class.**
* **Cross-laboratory collaborative data analyses, such as a Round-Robin tests, can be easily realized through generation of CHADA groups. The CHADA files included have to be calibrated beforehand, which can be done using an external (sample standard from WP3) or internal reference. In this case, calibrations for all instruments involved are generated by comparing a selected reference spectrum to spectra from the same sample recorded with the respective instrument.**
* **CHADA can be generated and used either locally (by installing *Python* and the CHADA library) or through web-based (browser) applications hosted by CHARISMA (WP5). In the latter case, no software installation is required on the user side.**

# **Appendix A: Python pseudocode for CHADA data transformer for background subtraction (chada.base)**

The following pseudocode plots a baseline-subtracted spectrum stored inside a CHADA file. First, a baseline model is generated using the previously defined function *baseline\_als*, implementing asymmetric least squares smoothing[[1]](#footnote-1) with the given parameters (*lamda, p, n\_iter*; *Method .fit*). The resulting baseline is stored as part of the transformer instance. This must only be done once, then the transformer is fitted. For baseline correction (*Method* .transform), the (fixed) baseline is subtracted from the spectrum being transformed. The use of a transformers list avoids storage of redundant data (original spectrum + baseline model + corrected spectrum). The helper function *readData* reads the *Native Data* and applies all transformers in the transformer list.

**# *CHADA baseline correction transformer is defined in library (updateable)***

**class ChadaBaselineTransformer():**

**self.type = “baseline”**

**def fit(self, spectrum\_y, parameters):**

**self.baseline\_model = baseline\_als(spectrum\_y, parameters)**

**def transform(self, spectrum\_k, spectrum\_y):**

**return spectrum\_k, (spectrum\_y - self.baseline\_model)**

**# *When a CHADA file is baseline-corrected, a transformer is constructed. Its parameters are written into the transformers list in the CHADA archive***

**def base(chada\_file\_name, parameters):**

**bt = ChadaBaselineTransformer()**

**bt.fit(readData(chada\_file\_name), parameters)**

**transformers = readTransformersList(chada\_file\_name)**

**transformers.append( {“type”: bt.type, “parameters”: bt. baseline\_model}**

**writeTransformersList(chada\_file\_name, transformers)**

**# *A spectrum in a CHADA file is plotted in its current transformation state***

**def readData(chada\_file\_name):**

**k, y = readNativeData(chada\_file\_name)**

**transformers = readTransformersList(chada\_file\_name)**

**for t\_info in transformers:**

**t = transformers\_dict[ t\_info[“type”] ]**

**k, y = t.transform(k, y, t\_info[“parameters”])**

**return k, y**

**def plot(chada\_file\_name):**

**plot( readData(chada\_file\_name) )**

# **Appendix B: Glossary**

***API: Application programming interface***

***Baseline Separation:* A numerical function that calculates a numerical model for a low-frequency baseline, which is superimposed on a Raman spectrum. The baseline is often assumed to be sample-independent and therefore sometimes regarded as an artifact. An example is the flourescent background generated by a number of materials. The baseline model is subtracted from the original spectrum, and stored separately.**

***Calibration***: Spectrum channel-dependent modifications of the spectrum *k* or *y* data, performed in order to correct for some instrument-dependent artifact. The aim of a calibration is to reduce or eliminate the instrumental influence on the spectral data, so that the latter only depends on sample properties. This makes Raman data comparable, even if it is recorded using different instruments and modes. The calibration model, i.e. the parameters for data modification on each channel, are usually generated by comparing standards with spectra recorded using the instrument to be calibrated. Examples are Raman Shifts (*k*) calibration and gain (*y*) calibration.

***Dynamic Metadata*: Metadata that changes during spectrum processing and conversion.**

***k: Raman Shifts***

***Method:* A numerical function that can be applied to Raman data. Examples are processing (e.g. normalization, interpolation on the Raman Shifts axis), visualization (e.g. spectrum plotting), or saving to disk.**

***Native Data:* The data and file format that is typically used for saving the data acquired with a specific Raman instrument. Often, a OEM-specific file format is used. The Raman data type (Raw, Primary etc.) depends on the file format and file saving protocol of the OEM software.**

***OEM:*** Original Equipment Manufacturer

***Primary Data:* Data that has already some level of mathematical processing to create a full Raman spectrum - e.g. pixel binding, stitching data segments, spectrometer offset correction or applying corrections that contaminate the spectrum, such as hot pixels. This data will be already displayed in cm-1 or Dcm-1.**

***Raman Shift:* The spectral energy shift with respect to the elastic Rayleigh peak, usually given in wavenumbers (cm-1).**

***Raw Data:* To be defined as Raman data as obtained from a calibrated and ready Spectrometer and prior to be submitted to any mathematical modification (stitching data, binding, correcting…) units in nm (x-axis) and arbitrary units (a.u., y-axis). Only corrections on the data, shall be the one associated with the spectrometer. i.e. the data will assign pixels readouts with nm as per calibration of the spectrometer and a.u. as intensity readout but will not be corrected in relationship of the incident monochromatic beam.**

***Static Metadata:* Metadata that remains unchanged during spectrum processing and conversion.**

***y:* Spectrometer counts**

1. Eilers, P. H., & Boelens, H. F. (2005). Baseline correction with asymmetric least squares smoothing. Leiden University Medical Centre Report, 1(1), 5. [↑](#footnote-ref-1)