# Description of the CHADA file format for Raman spectroscopy data developed as part of the CHARISMA Horizon 2020 project

Delivered by CHARISMA Work Package 4 (Deliverable 4.1)

Bastian Barton, Fraunhofer LBF

Version 1 - February 12, 2021

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

**CHARISMA proposal Objective 4: *Harmonise Raman characterisation data (Ref. WP: 4)***

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

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 characterisation in real industrial cases
4. Each CHADA file must contain a minimal set of metadata that will be defined and structured appropriately (T4.1)
5. CHADA must avoid information loss (preserve the original / native data)
6. The spectral resolution, i.e. the number of wavenumber channels, must be sufficient
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. **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 files (.cha)**
2. **CHADA groups (.chag)**
3. **CHADA calibration files (.chacal)**
4. **CHADA models (.chamod)**

**Furthermore, some functions will be needed that are not part of a CHADA class, e.g. for loading an existing CHADA file from disk, batch-generate CHADA files from a set of native Raman data files, or batch-transform a set of CHADA files (normalize, baseline-correct).**

* 1. **Structure of the CHADA file (.cha)**

**CHADA files are stored as pickled binary files containing a Python class instance. They contain the core Raman data and metadata and are meant to be permanently stored, shared and distributed. They are the basis for CHADA groups and databases and can be added to the latter. CHADA files consist of (cf.** Figure 1**):**

* **metadata (static and dynamic blocks;** Table 1**)**
* **a copy of the native data as binary**
* **a list of data transformers including a pointer to the current processing state, which can be moved and reset**
* **methods applicable to data and metadata (e.g. chada.plot, chada.export)**

**As data, a CHADA file can hold a single spectrum (0 dimensions plus wavenumbers), 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 sinlge 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.**

* 1. **Structure of the CHADA group file (.chag)**

**CHADA group files are stored as pickled binary files containing a Python class instance. 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. The set of constituting CHADA (.cha) files is chosen upon group initializaiton, and the current data – i.e. the (*k, y*) as resulting from the current processing state det by the .cha file pointer - is converted to a common wavenumber 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.**

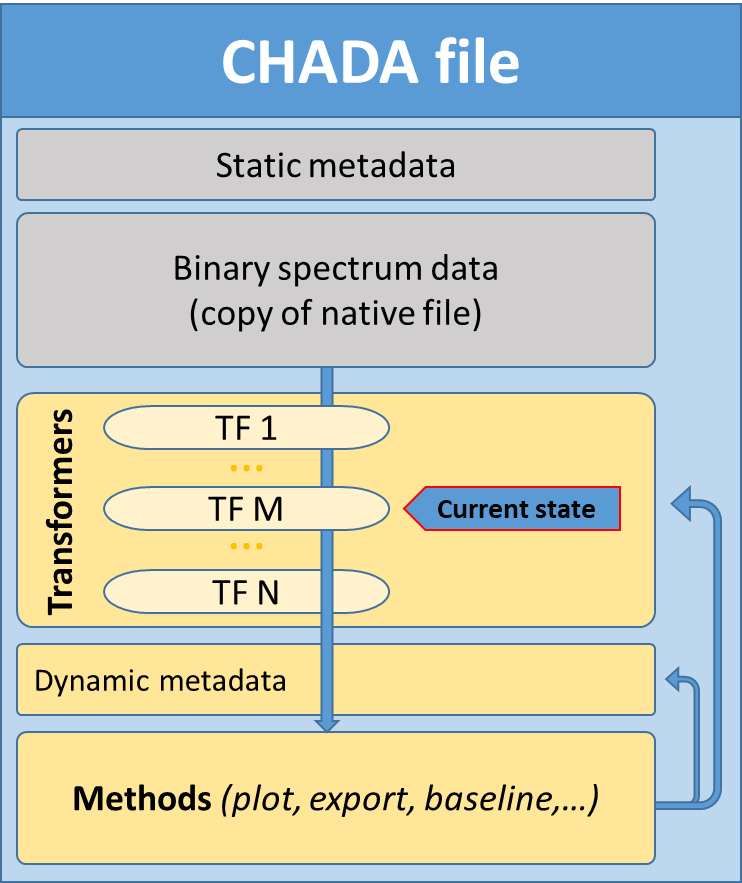
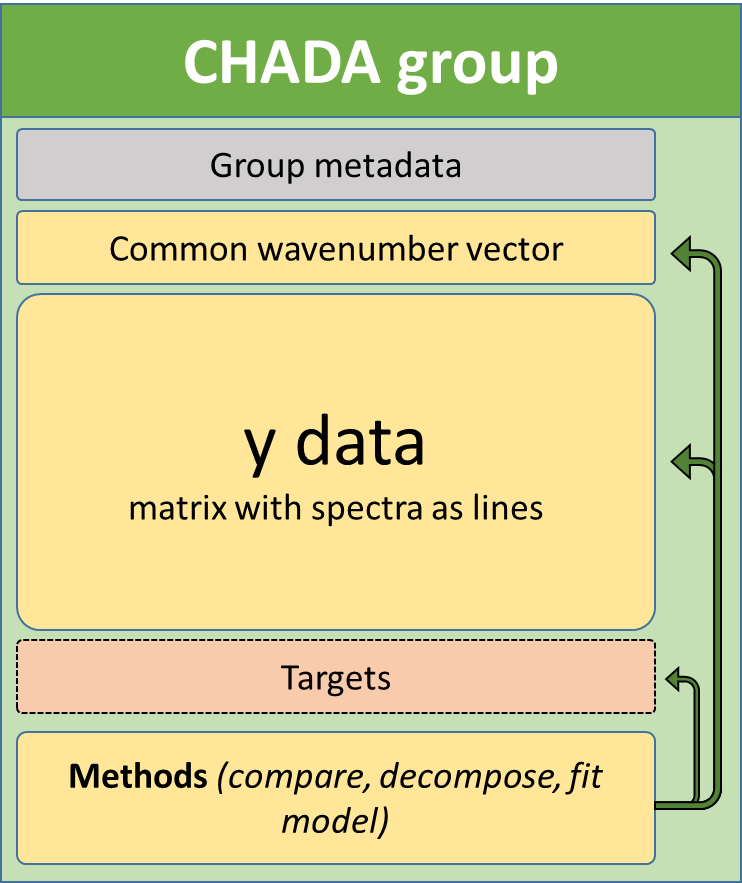
** **

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

* 1. **CHADA calibration files (.chacal)**

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

* **Wavenumber-dependent Raman shift calibration –> k axis**
* **Wavenumber-dependent gain (amplitude) calibration -> y axis**
* **Poit spread function deconvolution (PSF correction) -> y axis**
* **Modultion transfer function deconvolution (MTF correction) -> y axis**

**They 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 & mode can be calibrated using the corresponding .chacal file, and thereby become interoperable and comparable (**Figure 2**).**

* 1. **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 (wavenumber range, normalization, …) that must be applied to each CHADA file previous to prediction.

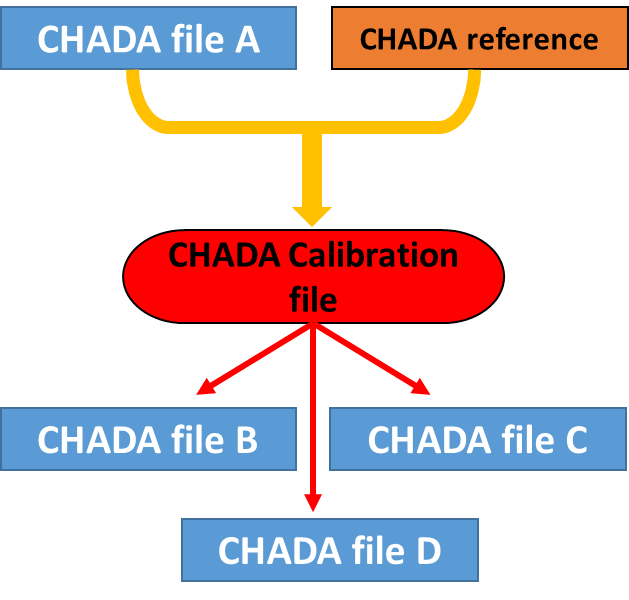
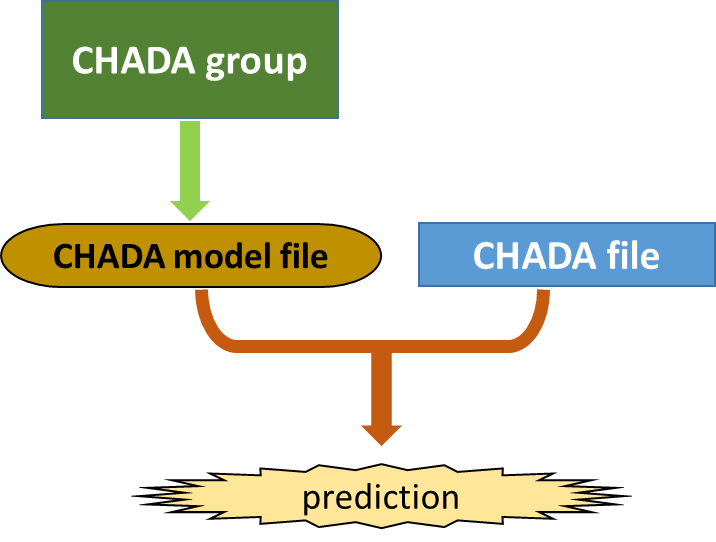
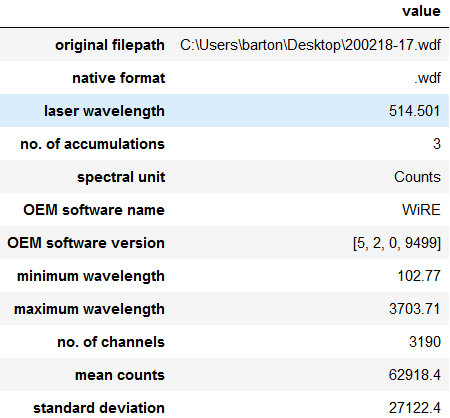
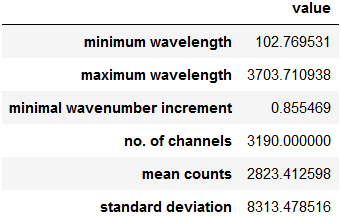
** **

Figure 2: Flow diagrams illustrating the generation and use of CHADA calibrations (left) and CHADA models (right). (left) A wavenumber calibration can be batch-applied to all CHADA data obtained from the same instrument / mode. 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 classifiy materials.

* 1. **Listing of CHADA file metadata**

Table 1: Examples for static (left) and dynamic metadata (right) included in a CHADA instance

* 1. **Steps for initialization of a .cha class instance**

1. **Read native file as binary and store as chada.binary**
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**
4. **Extract metadata from native metadata and *(k,y)* data, and store in chada.metadata()**
5. **Check if list of initial transformations has been given by user (e.g. “-b –s –c[310,1890]“ = baseline + smooth + crop wavenumbers to 310 – 1,890 1/cm)**
6. **Evoke methods corresponding to initial transformer list, generate dynamic metadata, and update transformer list**
   1. **Steps for initialization of a .chag class instance**
7. **Read specified set of CHADA files**
8. **Store metada of group members in *chag.metadatas***
9. **Add dimensionality of group members (0D, 1D, 2D, …) together with physical dimensions (map x/y in µm, time, …) in *chag.metadatas***
10. **From the dynamic metadata of the file set, find intersection of wavenumber vectors (only these channels are populated in all spectra and can thus be compared) -> set k\_min and k\_max for group wavenumber vector**
11. **Find minimal increment in k vectors of all files -> set increment for group wavenumber vector**
12. **Generate group wavenumber vector**
13. **Read all spectra from CHADA member files sequentially, interpolate to group wavenumber vector, and insert into *chag.y\_data* matrix as lines**
    1. **List of attributes of the CHADA (.cha) class**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| *'bands',* | Pandas DataFrame | Positions, amplitudes etc. for most prominent Raman bands found by the peaks() method. |
| *'binary\_data',* | binary stream | Native, unprocessed data and metadata |
| *'filename',* | string | Name of original file |
| *'metadata\_static',* | dictionary | metadata extracted from native 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 Raman native file extensions |
| *'transformers'* | list of CHADA transformer class instances | Fitted transformers corresponding to the processing steps that have been applied (e.g. chada.base()) |
| *'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 |

* 1. **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 |
| *'wavenumbers'* | 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. |

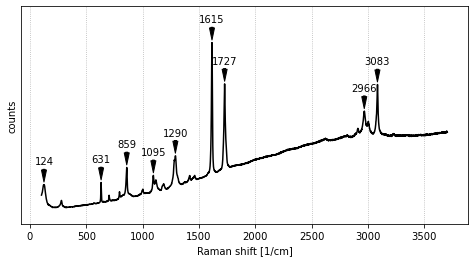
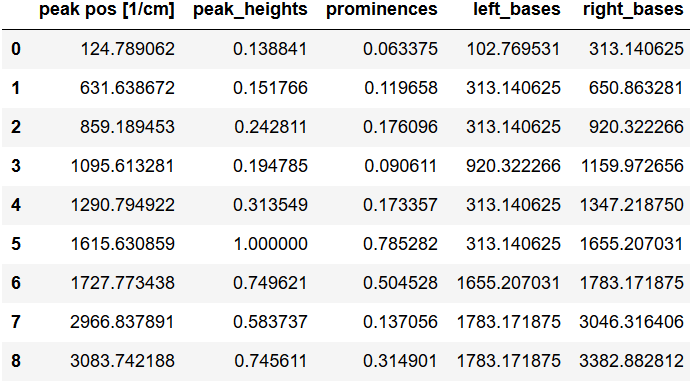
**** 

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 / CSV format.

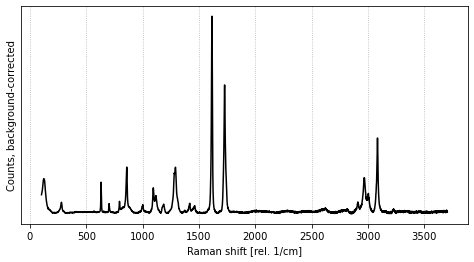
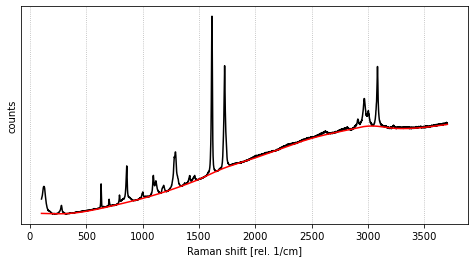


Figure 4: Demonstrator of the chada.base() method. A baseline model is fitted (left) and subtracted from the spectrum (right).

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 added to both the CHADA and CHADA group classes as needed according to the use cases.**

|  |  |
| --- | --- |
| **Use case** | **CHADA method(s)** |
|  |  |
| **Single** | |
| Single spectrum CHADA files contain native data, meta data, and transformers | |
| Create CHADA instance | *chada.\_\_init\_\_()* |
| 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()*** |
|  |  |
| **Group** | |
| CHADA groups interpolate CHADA spectra to a common wavenumber 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 (occurence 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 original, native data is always preserved as part of the CHADA (.cha) file container (requirement #5). It can be reproduced (dunped to disk) in its original format at any point unsing an appropriate method (*chada.native\_dump()*).**
* **The requirement of sufficient wavenumber range and sampling is always fulfilled, since CHADA works with transformers on the native data, rather than interpolating on a common wavenumber axis (requirements #6 & 7). An interpolation (data manupulation) is only performed when a CHADA group is created. In this case, data is interpolated on a k axis being the intersection of all group member axes, and the minimal increment occuring in all individual axes is used for sampling.**
* **No data is duplicated upon spectrum data processing. The size of singe CHADA files always stay 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 are implemented in 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 defines a transformer class for baseline subtraction. First, a baseline model is generated using the function baseline\_als, impementing 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 transformers avoids storage of redundant data (original spectrum + baseline model + corrected spectrum). The function *chada.data()* reads the original *(k,y)* data and applies all transformers in the transformer list up to the *chada.current\_state* pointer.**

**class ChadaBaselineTransformer():**

**def fit(self, spectrum\_ wavenumbers, spectrum\_counts, parameters):**

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

**def transform(self, spectrum\_ wavenumbers, spectrum\_counts):**

**return spectrum\_ wavenumbers, (spectrum\_counts - self.background\_model)**

**class Chada():**

**…**

**def base(self, parameters):**

**bt = ChadaBaselineTransformer()**

**bt.fit(self.data(), parameters)**

**self.transformers.append(bt)**

**…**

**def data(self):**

**wavenumbers, counts = self.read\_native\_data()**

**for t in self.transformers[:self.current\_state]:**

**wavenumbers, counts = t.transform(wavenumbers, counts)**

**return wavenumbers, counts**

**…**

**def plot(self):**

**plot(self.data())**

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)