Contents

1 Input script

pyProCT's input file is a human-readable JSON text file. Its file extension can be arbitrarily chosen by the user, but pyProCT generally produces this kind of files with a '.json' extension.

The input file describes how the software interacts with the underlying hardware where it is being executed, how the clustering exploration will be performed, and finally, which kind of postprocess operations will be performed after the clustering has been obtained. This section will be structured in 4 subsections, one for each of the main subsections of the script ('global', 'data', 'clustering' and 'postprocess').

Unless otherwise specified, each property or object will be represented by a descriptor. This descriptor contains the label of the property or JSON object (subsections), information about its data type, and possible dependencies.

A label consists of a double colon separated list of the names of the objects that encompass the property or subsection, followed by its name. For instance, **x::y::z** corresponds to the JSON object:

Listing 1: Simple JSON object.

```
"x":{
        "y":{
                "z":{}
        }
}
```

The nature of z will be specified after the label. Possible tags are:

Subsection	The item is another JSON object that can contain another subsections (JSON objects) and properties.
Integer	The item is an integer value property.
Real	The item is a real value property.
String	The item is a string property.
List	The item is an array of elements.

If the value of the property has to be chosen from a limited list of possible choices, this values will be enumerated in a "value in [choices]" clause. "Optional" will be added to the end of the descriptor if the property is optional.

Finally, if a property or subsection needs it, a dependency clause will be added in front of the label. This dependency clause contains the label it depends on and the value it needs to have, or, if it just depends on its previous definition, an "is defined" clause.

Examples:

[**x::y::z** is defined] **x::y::t**

Means that x::y::t value will be ignored unless x::y::z has a value.

[**x::y::z** == "m"] **x::y::t**

Means that x::y::t will not be used unless x::y::z is defined and has value "m".

1.1 global

1.1.1 control

global::control Subsection

Defines the type of scheduler used to run tasks (in the clustering generation and evaluation steps) and its parameters.

global::control::scheduler_type String, value in ['Process/Parallel', 'MPI/-Parallel', 'Serial']

It defines the type of scheduler that will be used in the clustering exploration and clustering evaluation steps. If set to 'Serial', one task starts when the previous one finishes. If set to 'Process/Parallel' a multiprocessing-based scheduler will be used, so multiple tasks can be executed at the same time, decreasing the total amount of time needed to perform that process. Finally, if 'MPI/Parallel' is used, an MPI-based scheduler will be built (this scheduler must be used only with pyProCTs MPI driver).

[global::control::scheduler_type == 'Process/Parallel'**]**
global::control::number_of_processes Integer

Indicates the number of processes that will be created to run the tasks if the 'Process/Parallel' scheduler was chosen. It defaults to *multiprocessing.cpu_count()* if the property is not defined. As the scheduler implements a master-slave scheme, this parameter has to be greater than 1, otherwise the scheduling will have undetermined behaviour (it will hang indefinitely).

Listing 2: This global section defines a "Process/Parallel" scheduler that uses 4 processes.

```
"control": {
        "scheduler_type": "Process/Parallel",
        "number_of_processes": 4
}
```

1.1.2 workspace

In this section the paths used by pyProCT for this project will be defined. Only workspace::base property is mandatory. If the other properties are not defined they will default to "results", "tmp", "matrix" and "clusters" respectively.

global::workspace::base String

Path of the workspace folder for this project. Mandatory.

global::workspace::results String, Optional

Name of the folder inside the project workspace where results are to be stored.

global::workspace::tmp String, Optional

Name of the folder inside the project workspace used to store temporary files.

global::workspace::matrix String, Optional

Name of the folder inside the project workspace used to store the distance matrix (if required).

global::workspace::clusters String, Optional

Name of the folder inside the project workspace where cluster-related files (if generated) will be stored. Currently all the clusterings with their clusters will be stored as part of the results file.

Listing 3: Complete workspace section for a project called ClusteringProject in John's home folder

```
"workspace": {
        "tmp": "tmp",
        "matrix": "matrix",
        "clusterings": "clusterings",
        "results": "results",
        "base": "/home/john/ClusteringProject"
}
```

1.2 data

data::type String, Optional

Not used property.

1.2.1 matrix

In this section the user will define how the distance matrix is generated.

data::matrix::method String, value in ['load', 'rmsd', 'distance']

Defines the method used to generate the distance matrix.

With 'load' pyProCT will try to load a previously created matrix instead of generating one.

With 'rmsd' it will calculate a pairwise RMSD matrix. It can calculate the RMSD of the whole system, or parts of it if a different selection was used to superpose.

Finally, with 'distance' it will calculate the euclidean distance matrix of part of the system (for instance, in a ligand-receptor system, one would superpose receptors and calculate the pairwise distance of ligands in order to cluster them).

data::matrix::filename String, Optional

If defined, the distance matrix will be saved in a file named like this, inside the folder defined in workspace::matrix.

data::matrix::image Subsection, Optional

If defined, an image representation of the distance matrix will be rendered.

data::matrix::image::filename String

Complete path of the image file to be generated, without extension.

data::matrix::image::dimension String, Optional

Leading dimension of the matrix plot in pixels. Defaults to 1000 pixels.

data::matrix::parameters Subsection

The parameters of the selected method will be defined here.

data::matrix::parameters::calculator_type String

One of the available calculators available in the local pyRMSD installation.

[data::matrix::method == 'rmsd' or 'distance'**]**

data::matrix::parameters::fit_selection String

A Prody[**?**]-compatible selection string defining the part of the system that will be used for fitting (superposition).

[data::matrix::method == 'rmsd'**]**

data::matrix::parameters::calc_selection String, Optional

A Prody-compatible selection string defining the part of the system that will be used for RMSD calculation.

[data::matrix::method == 'distance'**]**

data::matrix::parameters::body_selection String, Optional

A Prody-compatible selection string defining the part of the system that will be used to calculate the euclidean distances.

[data::matrix::method == 'load'**]**

data::matrix::parameters::path String

Path of the matrix file to load without extension (absolute or relative to the execution path).

Listing 4: Loading a distance matrix from a previous clustering project to save calculation time.

```
"matrix": {
        "method": "load",
        "parameters": {
                "path":"/home/john/ClusteringProject/matrix/mymatrix"
        }
}
```

Listing 5: Generation of a pairwise RMSD distance matrix using pyRMSD's QCP calculator in its OpenMP version. Only alpha carbons will be used to superpose and calculate RMSD

```
"matrix": {
```

```
          "method": "rmsd",
          "parameters": {
                  "calculator_type": "QCP_OMP_CALCULATOR",
                  "fit_selection": "name CA"
          }
}
```

Listing 6: Generation of a pairwise euclidean distance distance matrix. Alpha carbons will be superposed and the ligand's geometric center, selected with the body_selection tag, will be used to calculate distances.

```
"matrix": {
          "method": "rmsd",
          "parameters": {
                  "calculator_type": "QCP_OMP_CALCULATOR",
                  "fit_selection": "name CA" ,
                  "body_selection": "resnum 530 not name H1 H2 H3 HB2 HB3 HG2 H
          }
}
```

data::files List(String)

A list with the paths of the pdbs containing the conformations that will be clustered.

Listing 7: Loading two pdb files called A and B, residing in the home directory of user 'john'

```
data:{
...,
          files:[
                  /home/john/A.pdb,
                  /home/john/B.pdb
          ]
}
```

Both A and B must have the same number of atoms except if the selections applied to them produce the same number of atoms and there exists a naive one to one mapping between them e.g. 'A.pdb' is an all-atom model, 'B.pdb' a CA model obtained from A and only CAs are used for superposition.

data::matrix::parameters::symmetries Subsection, Optional

Contains named symmetry objects that define the calculation symmetries. It is useful to define atom correspondences so that flips can correctly be handled.

A symmetry object consists of a common selection part, and a list of equivalences. Each equivalence is a pair of 1 atom selections that can be exchanged. All equivalences in the same list are treated at the same time, however equivalences for different objects will be recursively applied in a way that all combinations of atom exchanges are calculated.

Symmetry handling is still under development.

1.3 clustering

The clustering section contains the subsections and parameters needed to specify the details of both the clustering exploration and evaluation (and thus the description of the clustering hypothesis).

1.3.1 clustering exploration

clustering::generation Subsection

Contains details regarding the obtention of the best clustering.

clustering::generation::method String, value in ['load', 'generate']

Defines the method used to obtain the best clustering. If 'generate' is chosen, it will perform the clustering exploration protocol. If 'load' is used instead, it will use the clustering json object placed in the clustering::generation::clusters section.

[clustering::generation::method == 'load'**] clustering::generation::clusters** List(ClusterObject)

Defines the clustering that will be loaded if the generation method is 'load'. A cluster object contains 3 properties:

id	Is a unique identifier for the cluster.
prototype	The element of the dataset that is the medoid of the cluster.
elements	A string containing a human-readable compressed description of the elements of the dataset forming part of this cluster. This string is comprised of a series of comma-separated integers and integer ranges. A range is formed by 2 integers joined by a colon symbol (:). It represents the number of elements that lie from the initial integer to the last (both included). Ranges of elements are preferred over single elements. Single elements will only be written when the size of the range is 1.

Listing 8: Loading a clustering consisting on 2 clusters. The dataset has 22 elements. 'cluster_00' has 8 elements and 'cluster_01' has 14 elements.

```
"clustering": {
        "generation": {
        "method":"load",
        "clusters": [
                        {
                                "prototype": 16,
                                "id": "cluster_00",
                                "elements": "9, 14:20"
                        },
                        {
                                "prototype": 7,
                                "id": "cluster_01",
                                "elements": "0:8, 10:14, 21"
                        }
                ]
        }
}
```

clustering::algorithms Subsection

All algorithms share 2 properties: 'max' and 'parameters'. An algorithm must be explicitly added to the algorithms subsection. There exists one entry per implemented algorithm. Currently there are 6 implemented algorithms with keywords: 'dbscan', 'gromos', 'hierarchical', 'kmedoids', 'random' and 'spectral'.

clustering::algorithms::ALGORITHM::max Integer, Optional

Máximum number of clusterings that can be generated using this algorithm.

clustering::algorithms::ALGORITHM::parameters List(ParameterObject), Optional

If present, contains a list of ParameterObject elements, each of them storing a different parametrization of the algorithm. If this section is not defined, pyProCT will try to automatically guess the best parameters to be used in the exploration for that algorithm.

[clustering::algorithms::ALGORITHM::parameters is defined**]**

clustering::algorithms::ALGORITHM::max Integer, Optional

Maximum number of parameter sets that pyProCT will generate.

clustering::algorithms::kmedoids::seeding_type String, value in ['RANDOM', 'EQUIDISTANT'], Optional

Special property of k-medoids. Can have two values:

'RANDOM' – Initial seeds will be randomly placed.

'EQUIDISTANT' – Initial seeds will be placed at equidistant points (it treats the ensemble as a sequence). This is the default value.

[clustering::algorithms::kmedoids::seeding_type =='RANDOM'**]**

clustering::algorithms::kmedoids::tries Integer, Optional

Defines de number of times the algorithm will be repeated using a different seed. If not defined, it defaults to 10.

clustering::algorithms::spectral::sigma Real, Optional

Special property of spectral clustering. It defines the value of the sigma parameter that will be used for all executions of the algorithm (defining more than one value of sigma is not allowed because of performance reasons). If it is not present, the default action is to calculate local sigmas.

1.3.2 Evaluation

Contains subsections and properties to express the clustering hypothesis.

clustering::evaluation::minimum_clusters Integer

All clusterings with less clusters than this number will be filtered out.

clustering::evaluation::maximum_clusters Integer

All clusterings with more clusters than this will also be ignored in the evaluation step.

clustering::evaluation::minimum_cluster_size Integer

Any cluster with a size smaller than minimum_cluster_size will be considered a noisy cluster. All its elements will be removed from the clustering and the noise of this clustering will increase.

clustering::evaluation::minimum_noise Real

Any clustering with a noise percentage higher than this will be discarded.

clustering::evaluation::query_types List(String)

A list clustering property and quality function types.

Listing 9: Defining a list of 3 clustering properties.

```
"query_types": [
        "NumClusters",
        "NoiseLevel",
        "MeanClusterSize"
]
```

clustering::evaluation::evaluation_criteria Subsection

This subsection contains a series of criteria objects. A criteria object contains one or more evaluation objects. Each evaluation object is defined by:

- The name of a quality function.
- An action property: '<' if the value has to be minimized, '>' if it has to be minimized. For instance, one would like to maximize GaussianSeparation (maximize clustering separation) and minimize PCAanalysis (minimizing cluster variance, which would mean maximizing cluster compactness).
- A relative weight for that quality function. It is not mandatory that all weights are smaller than 1 or that they sum 1.

Example 4: A CythonSilhouette evaluation object with weight 0.4 which value will be maximized.

Listing 10: A CythonSilhouette evaluation object with weight 0.4 which value will be maximized.

```
"CythonSilhouette": {
"action": ">",
"weight": 0.4
}
```

A criteria object is defined by a unique name and its evaluation objects. All criteria will be evaluated independently. The best clustering will be the clustering with a better score in one of them.

Example 2: Suppose that after the filtering step we have only 2 clustering candidates A, B and C and 2 criteria. All criteria will be evaluated for each cluster:

	Evaluation (ICV)	clustering_A	clustering_B	clustering_C
criteria_0	CythonSilhouette	11	45	10
	Dunn	0.5	0.4	0.23
criteria_1	GaussianSeparation	3	2	1

After min-max normalization:

	Evaluation (ICV)	clustering_A	clustering_B	clustering_C
criteria_0	CythonSilhouette	0.03	1	0
	Dunn	1	0.63	0
criteria_1	GaussianSeparation	1	0.5	0

In this case all evaluation objects are equall weighted and its values will be maximized. The final criteria scores will be as follows:

	clustering_A	clustering_B	clustering_C
criteria_0	0.5	0.82	0
criteria_1	1	0.5	0

Note that final score values are always in the 0 to 1 range, so the best clustering will be the one with the best score for any of the defined criteria. In this case, the best clustering would be 'clustering_A' for criteria_1.

If no criteria is defined in the evaluation section, then the best clustering is chosen randomly between the non-filtered clusterings.

Listing 11: A complete evaluation section to ensure that the selected best clustering will have at most a 10% of noise, a number of clusters in the range 1-50 and that every cluster of that clustering will contain at least 60 elements. Some queries and 2 criteria are also defined

```
"evaluation": {
        "maximum_noise": 10,
        "minimum_cluster_size": 60,
        "maximum_clusters": 50,
        "minimum_clusters": 2,
        "query_types": [
                "NumClusters",
                "NoiseLevel",
                "MeanClusterSize"
        ],
        "evaluation_criteria": {
                "criteria_0": {
                        "CythonSilhouette": {
                                "action": ">",
                                "weight": 0.4
                        },
                        "PCAanalysis": {
                                "action": "<",
                                "weight": 0.3
                        },
                },
                criteria_1:{
                        "GaussianSeparation":{
                                "action": ">",
                                "weight": 0.3
                        }
                }
        }
}
```

1.4 Postprocess

In this section users will define the postprocess actions that will be performed using the resulting clustering. Postprocess is the only optional main section.

postprocess::rmsf Subsection

If defined, pyProCT will generate an rmsf file containing global and per-cluster rmsf data for plotting. It uses the selections. It can be viewed with the results viewer.

postprocess::centers_and_trace Subsection

If defined, pyProCT will generate a data file with the backbone trace of the biomolecule and the centers of mass of the selections. It can be viewed with the results viewer.

postprocess::representatives Subsection

If defined, pyProCT will generate a pdb file containing the medoid of each cluster of the best clustering.

postprocess::representatives::keep_remarks Boolean, Optional

If true, the remark lines before a pdb model section will be extracted along with the model . If not defined, it defaults to false.

postprocess::representatives::keep_frame_number Boolean, Optional

If true, the 'model number' of each conformation will be renumbered so that the first model of each file will be "model 1", the second "model 2" and so on.

postprocess::pdb_clusters Subsection

If defined, pyProCT will generate a compressed file containing a pdb file for each cluster. This pdb files will contain the conformations the cluster holds.

postprocess::pdb_clusters::keep_remarks Boolean, Optional

If true, remark lines before any conformation will be extracted along with the conformation. If not defined, it defaults to false.

postprocess::pdb_clusters::keep_frame_number Boolean, Optional

If true, the 'model number' of each conformation will be renumbered so that the first model of each file will be "model 1", the second will be "model 2" and so on.

postprocess::compression Subsection

If defined, pyProCT will produce a compressed version of the input trajectories.

postprocess::compression::final_number_of_frames Integer

The final number of frames that the output trajectory must have.

postprocess::compression::file String name of the output file ("compressed.pdb" by default)

postprocess::compression::type String, value in ['RANDOM', 'KMEDOIDS']

Method used to compress. If 'RANDOM' is used, the elements of the compressed dataset will be randomly sampled. If 'KMEDOIDS' is selected, the element of the compressed dataset will be created by partitioning each cluster using the k-medoids algorithm and selecting the medoid of each of these subclusters.