DATABASE

leaving aside for a while other fields, as Computers, Code or others, the main two objects are Data and Calc

class Calc

class Data

↪ typename = ‘...’ # to understands if it represents a structure, a pseudo, params...

* + TypeName:

StructData

ParamData

…

Properties:

* + UUID
  + TypeName # from which we derive the python subclass
  + Date (creation/modification)
  + jsons = { }
  + files = [ ]
  + ‘extra’ field? # It could be used to help in building ordered correspondance between atoms and pseudos

Methods:

* + def \_add\_json()
  + def \_del\_json()
  + def \_list\_jsons()
  + def \_get\_json()
  + def \_add file()
  + def \_del\_file()
  + def \_list\_files()
  + def \_get\_file() # to decide if filename or fileobject
  + attach\_backend() # associate a database backend to the DataObject, to be defined in the constructor of Data ?
  + store()

Example of use:

b = Backend(type = Filesystem, location = /tmp)

d = Data(backend = b, Type = pseudo, )

d.functional = ‘LDA’

d.element = ‘C’

d.store() # (will call the store\_file and store\_json methods of the backend)

low level operations: # be careful at how not to lose data

d.validate()

b.save\_flies()

b.save\_jsons()

b.type()

b.uuid()

* + get\_url / uri ?

Every type will be a subclass and implement, specific properties and methods and will call the low level functions described above

Some subclasses:

* + SimpleFileData # a single file stored in the permanent repository

Properties:

* + - Files = [ ] # one single file
    - Jsons = { }

Methods:

* + - get\_content
    - get\_path # path on disk
    - ? get\_resource ? # for remote stuff
    - get\_md5
  + ? DirectoryData ?
  + RemoteFileData # a file stored on a remote computer, which can disappear, used for dependencies, eg scf-> phonons

Properties:

* + - files = [ ]
    - jsons = { url = } # the json consists only in the info of url

Methods:

* + - get\_url() # remote://resourcename/path\_relative\_to\_aida or rel
  + RemoteFolderData # as above for the whole folder

Can it become the same of RemoteFileData

* + StructureData # a cell data

Look more or less at the class already implemented by Giovanni

* + PseudopotentialData # to understand also how to support Vasp

Or instead, for simplicity, just stick with

UpfData:

Properties:

* + - * element
      * functional
      * pseudo\_type
      * jsons = { upf\_info } # with a link to a json with more exaustive info on the pseudopotential

Methods:

* + - * get\_upf\_path()
      * content()
      * parse\_content()
  + ParameterData # will store a single json

Properties:

params = { }

Methods:

For the store and retrieve, use the same strategy used by Django to save elements in the Table, that is,

UUID = None, create a new object

UUID = num that exists: overwrite an existing object

UUID = num that doesn’t exist: think at something (raise exception? or write a new object?)

class Backend

Methods:

* + def store\_file()
  + def store\_json()

Properties:

* + Type # form which the subclasses are derived
  + Location
  + Password
  + … (similar to the configurations of django databases)

class Code:

Properties:

* + code\_family
  + version
  + default\_plugin
  + machine # can be None for a script or the uuid
  + exec\_abs\_path

Methods:

* + trivials to set/get the above params

class Calc:

Properties:

* + plugin(default = Code.DefaultPlugin)
  + code
  + in\_datas # a list of uuids?
  + out\_datas # a single uuid

Methods:

* + save()
  + set\_backend()
  + get\_status() # return CalcStatus, for the queuing status
  + get\_out\_datas # return the object Data output
  + set\_out\_data # pass an object Data and its name, to set it up as the outdata
  + get\_in\_datas # return the object Data output
  + set\_in\_data(namedata, DataObj)

Example

c = Calc()

s = StructData()

c.aset\_inparam(‘input\_cell’,s)

through table to link Indata and calc:

|  |  |  |  |
| --- | --- | --- | --- |
| calc | data\_uuid | name (may not be unique, eg if you need several pseudo) |  |
| c.1 | d.1 | cell |  |
| c.1 | d.2 | pseudo |  |
| c.1 | d.1 | in\_params |  |
|  |  | pseudo |  |

Indata are saved as properties of calc, while the outdata are saved as incalc nel dato figlio. Ogni dato c’ha un dato figlio o un parent. Da usare come json? dipende dal backend

class CalcStatus:

Properties:

* + statusname

To decide the possible values of statusname

Methods:

* + def get\_status()
  + def set\_status()