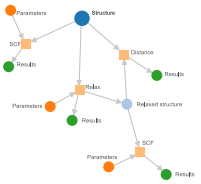
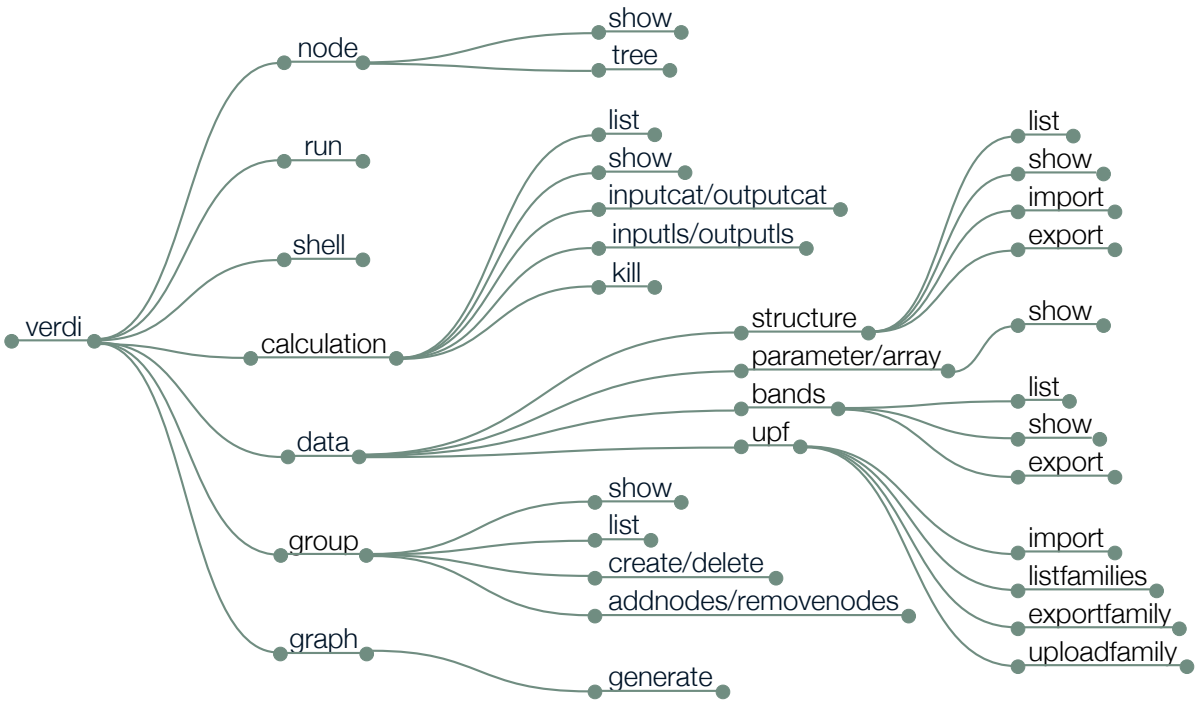
AiiDA



The `verdi` command-line tool

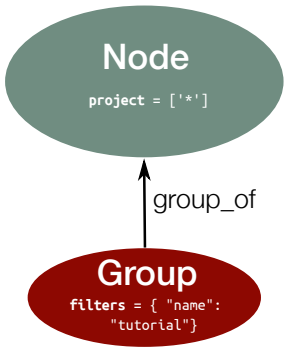
Use TAB auto-completion at any level for full list of commands and subcommands
Note that this is a non-complete list of commands, but it only summarizes the most common ones



The QueryBuilder

To import: `from aiida.orm.querybuilder import QueryBuilder`

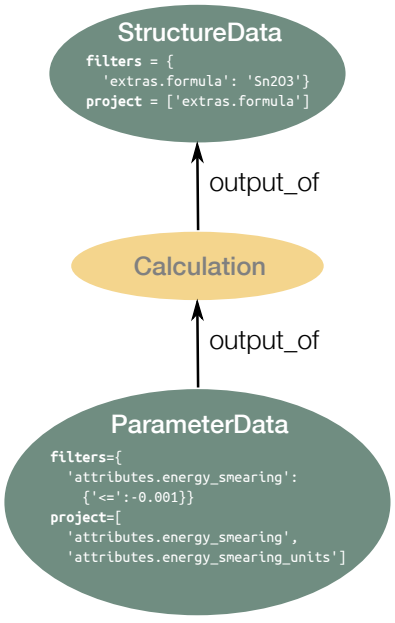
Fetch all nodes of group "tutorial"




```
qb = QueryBuilder()
qb.append(Node,
          tag="nodes",
          project=["*"])

qb.append(Group,
          group_of="nodes",
          filters={"name": "tutorial"})

qb.all()
```

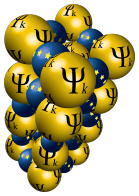
Print the smearing energy calculated for Sn_2O_3 if it is smaller than 10^{-4} eV

```
qb = QueryBuilder()
qb.append(
    StructureData,
    project=["extras.formula"],
    filters={"extras.formula":"Sn2O3"},
    tag="structure")

qb.append(
    Calculation,
    tag="calculation",
    output_of="structure")

qb.append(
    ParameterData,
    tag="results",
    filters={"attributes.energy_smearing":
    {"<=":-0.0001}},
    project=[
        "attributes.energy_smearing",
        "attributes.energy_smearing_units",
    ],
    output_of="calculation")

qb.all()
```



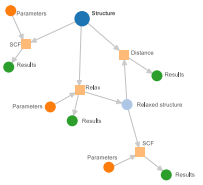
MARVEL
NATIONAL CENTRE OF COMPETENCE IN RESEARCH

MAX DRIVING THE EXASCALE TRANSITION

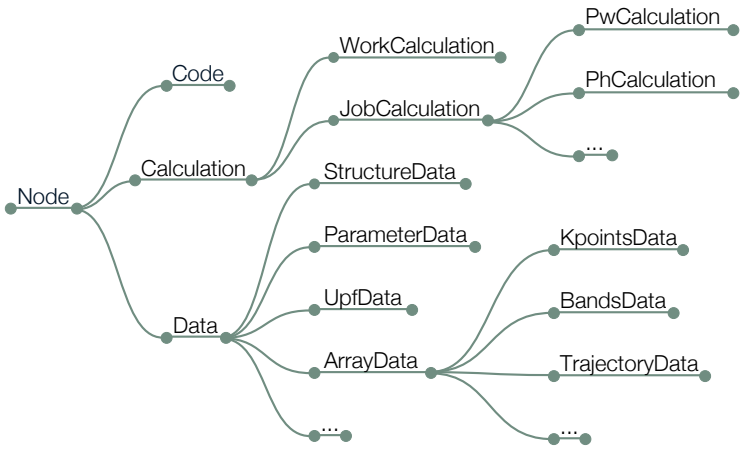


v18.05

The AiiDA cheatsheet



The main AiiDA Node subclasses



To load an existing node: `load_node(<pk>)` or `load_node(<uuid>)`

To load a class, either import it from `aiida.orm` or use the `DataFactory` (returning Data subclasses) or the `CalculationFactory` (returning JobCalculation subclasses)

The Factories

Naming conventions

Subclasses defined by AiiDA follow the convention described below.
In the following, **<fullname>** is a lower-caps, dot-separated string, while **<Name>** is just the final part after the last dot, capitalized.

If you would like to do
`from aiida.orm.data.<fullname> import <Name>Data`
you can simply do
`<Name>Data = DataFactory("<fullname>")`

Examples of **<fullname>**:
`"upf", "array", "array.kpoints", ...`

If you would like to do
`from aiida.orm.calculation.job.<fullname> import <Name>Data`
you can simply do
`<Name>Calculation = CalculationFactory ("<fullname>")`

Examples of **<fullname>**:
`"quantumespresso.pw", "quantumespresso.ph", ...`

Main attributes and methods

Note: each derived class inherits all the methods of the parent class

Node

`pk`	Node ID
`label`	Short label
`uuid`	Unique ID
`ctime`	Creation time
`mtime`	Modification time
`folder`	Repository folder
`inp.<linkname>`	Input node
`out.<linkname>`	Output node
`get_inputs()`	All inputs
`get_outputs()`	All outputs
`get_attrs()`	Querable attributes
`get_attr(k)`	Attribute 'k'
`get_extras()`	Querable extras
`get_extra(<k>)`	Extra 'k'
`set_extra(<k>,<v>)`	Set extra k=v
`get_comments()`	All comments
`add_comment()`	Add comment
`store()`	Save node in DB
`store_all()`	Save node+parents

Code

`get_from_string(<n>)`	Load code with n="name@machine"
`new_calc()`	Return new calc using this code

Data

`export()`	Export to file
`_exportstring()`	Export to string
`importfile()`	Import from file
`importstring()`	Import from string

StructureData

`cell`	Lattice vectors
`sites`	Atomic sites
`kinds`	Species with masses, symbols, ...
`pbc`	Periodic bound. cond. along each axis
`get_formula()`	Chemical formula
`get_cell_volume()`	Compute cell volume
`convert(<fmt>)`	Convert to ASE, pymatgen, ...
`set_cell(<c>)`	Set lattice vectors
`set_pbc(<pbc>)`	Set PBC
`set_ase(<a>)`	Create cell from ASE
`set_pymatgen(<p>)`	Create cell from pymatgen
`append_atom(<p>, <symb>)`	Add atom of type 'symb' at position 'p'

ParameterData

`dict.<k>`	Get value for key 'k'
`keys()`	Get all keys
`get_dict()`	Get all key/values
`set_dict(<dict>)`	Replace all key/values

ArrayData

`get_arraynames()`	Names of all arrays
`get_array(<n>)`	Get array 'n'
`set_array(<n>,<a>)`	Set/store array 'a' with name 'n'

KpointsData

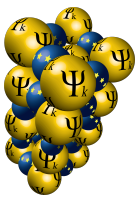
`set_kpoints(<k>)`	Set an explicit list of kpoints 'k' (optionally with weights)
`get_kpoints()`	Get explicit list of kpts (if stored explicitly)
`set_kpoints_mesh(<m>)`	Set an implicit mesh (e.g. 'm'=3x2x5)
`get_kpoints_mesh()`	Get the implicit mesh (if stored implicitly)

JobCalculation

`get_state()`	Calculation state
`get_computer()`	Computer where it is running
`get_code()`	Code used to run
`get_job_id()`	Scheduler job ID
`set_resources()`	Get # nodes, MPI procs per node, ...
`res.<k>`	Value of parsed output 'k'
`submit_test()`	Fake submit, just generate files
`submit()`	Submit calculation
`kill()`	Kill job on scheduler
`use_<xxx>(<y>)`	Set link from node 'y' as input of type 'xxx'
`_use_methods().key`	Dictionary of valid use_<xxx> methods, expected types of 'y', linkname used in the database, docs, ...

MARVEL
NATIONAL CENTRE OF COMPETENCE IN RESEARCH

MAX DRIVING THE EXASCALE TRANSITION



v18.05