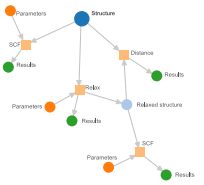
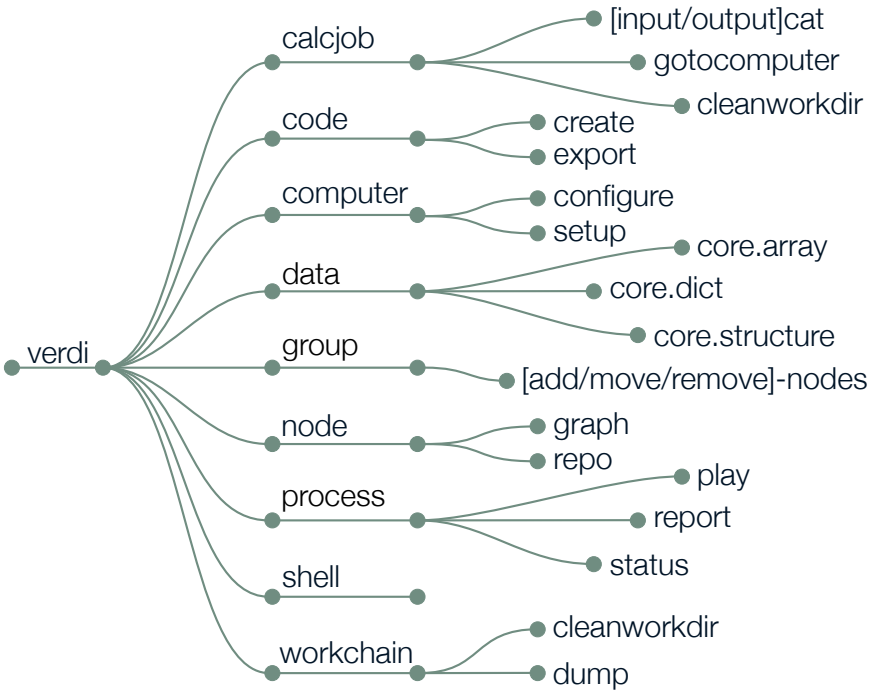
The AiiDA v2.5.1 cheat sheet



The `verdi` command-line API*



*Not exhaustive

*Most options also implement `show/list/delete`

Tools of the trade

Other `verdi` tips and tricks

Know what's there:

```
$ verdi profile list
$ verdi user list
$ verdi storage list
$ verdi plugin list aiida.calculations
$ verdi plugin list aiida.workflows
```

AiiDA to classical file tree:

```
$ verdi calcjob dump <pk>
$ verdi workchain dump <pk>
```

Config options, e.g. caching:

```
$ verdi config list
$ verdi config set \
    caching.default_enabled true
$ verdi config set caching.enabled_for \
    aiida.calculations:quantumespresso.pw
```

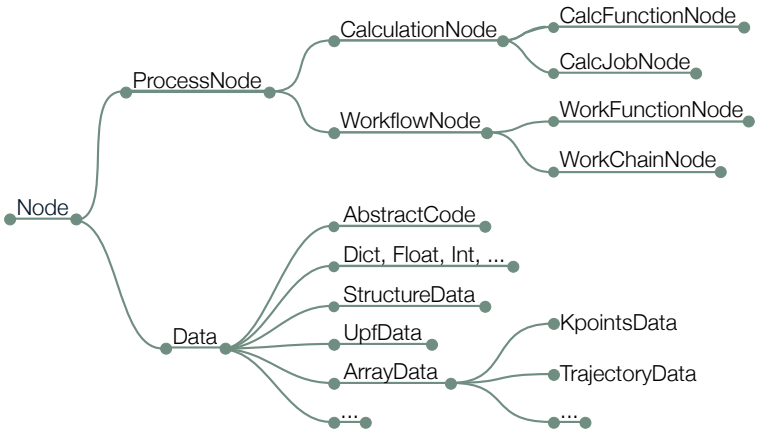
Fix what went astray:

```
$ verdi daemon restart --reset
$ verdi devel rabbitmq tasks analyze --fix
```

Share your data:

```
$ verdi archive create <.aiida-archive> \
    --groups/--nodes <groups/nodes>
$ verdi archive import <.aiida-archive>
```

The AiiDA Node subclasses



Additional web resources (click me)

`aiidalab` `aiida-project` `aiida-shell` `aiida-submission-controller`
`aiida-resource-registry` `aiida-tutorials` `aiida-plugin-cutter`

AiiDA Python imports

ORM, nodes, and Factories

Import aiida-core Node classes from aiida.orm:

```
from aiida.orm import Dict, CalcJobNode
```

Import Nodes via pk, UUID, or label:

```
my_node = load_node(<identifier>)
```

Import Data classes via the DataFactory:
(Note: Prefix AiiDA core types with `core`)

```
my_kpts = DataFactory("core.array.kpoints")
```

Import CalcJob classes via the CalculationFactory:

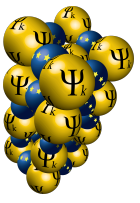
```
my_calcjob = CalculationFactory(
    "quantumespresso.pw"
)
```

Import WorkChain classes via the WorkflowFactory.

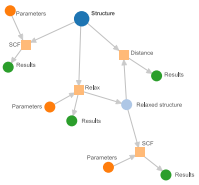
```
my_workflow = WorkflowFactory(
    "quantumespresso.pw.bands"
)
```

MARVEL
NATIONAL CENTRE OF COMPETENCE IN RESEARCH

MAX DRIVING THE EXASCALE TRANSITION



The AiiDA v2.5.1 cheat sheet



Main attributes and methods

Node

`pk`	Node ID
`label`	Short label
`uuid`	Unique ID
`ctime`	Creation time
`mtime`	Modification time
`get_incoming()`	Get input
`get_outgoing()`	Get output
`inputs`	All inputs generator
`outputs`	All outputs generator
`attributes`	Queryable attributes
`get_attribute(k)`	Attribute 'k'
`extras`	Queryable extras
`get_extra(<k>)`	Extra 'k'
`set_extra(<k>,<v>)`	Set extra k = v
`get_comments()`	All comments
`add_comment(<c>)`	Add comment with content <c>
`store()`	Save node in DB

Code

`load_code(<id>)`	Load code using `pk`, `UUID`, or `label`
`get_builder()`	Return new builder using this code

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_ase(<a>)`	Create cell from ASE
`set_pymatgen(<p>)`	Create cell from pymatgen
`append_atom(symbols=<symb>, position=<p>)`	Add atom of type 'symb' at position 'p'

Dict

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

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)

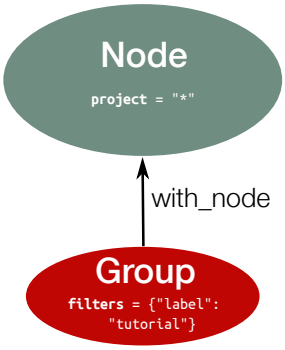
CalcJobNode

`process_state`	Calc. process state
`exit_status`	Exit status or int code
`is_finished`	Has calc. finished?
`is_failed`	Has calc. failed?
`computer`	Computer where it is running
`inputs.code`	Code used to run
`get_job_id()`	Scheduler job ID
`get_options()`	Get # machines, MPI procs per machine, ...
`res.<k>`	Value of parsed output 'k'

The QueryBuilder

To import: `from aiida.orm import QueryBuilder`

Fetch all nodes of group "tutorial"




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

qb.append(Group,
          with_node="nodes",
          filters={"label": "tutorial"})

qb.all()
```

Print the smearing energy calculated for BaO_3Ti if it is smaller than 10^{-4} eV

```
qb = QueryBuilder()
qb.append(
    StructureData,
    project=["extras.formula"],
    filters={"extras.formula":"BaO3Ti"},
    tag="structure")

qb.append(
    CalcJobNode,
    tag="calculation",
    with_incoming="structure")

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

qb.all()
```