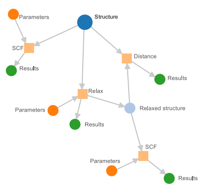
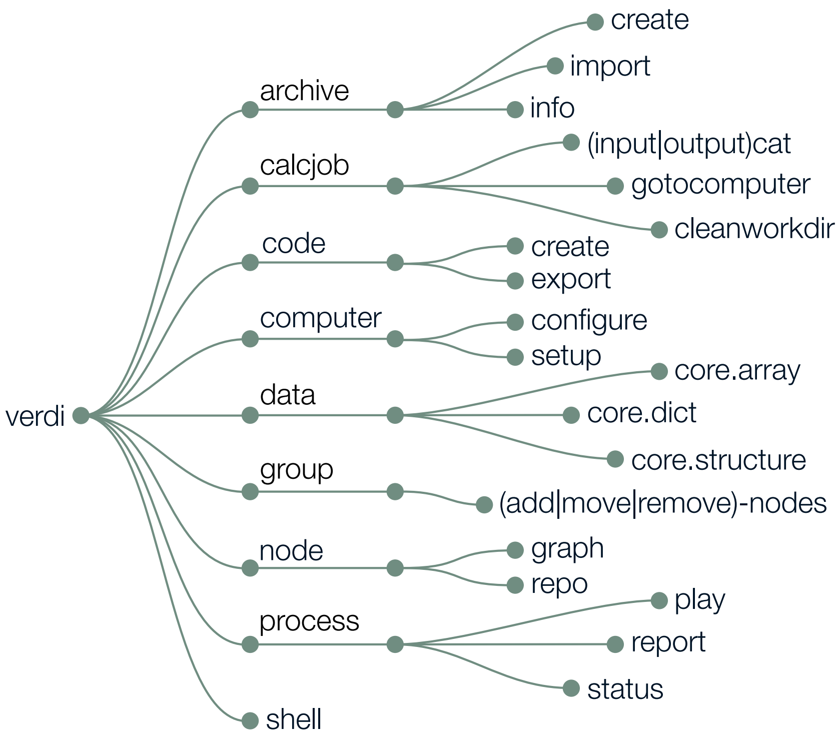
The AiiDA cheat sheet



The `verdi` command-line API*



*Not exhaustive
*Most options also implement `show/list/delete`

Tools of the trade

Other `verdi` tips and tricks

Quickstart:
```
$ verdi presto
```

Know what's there:
```
$ verdi profile list
$ verdi plugin list aiida.calculations
$ verdi plugin list aiida.workflows
```

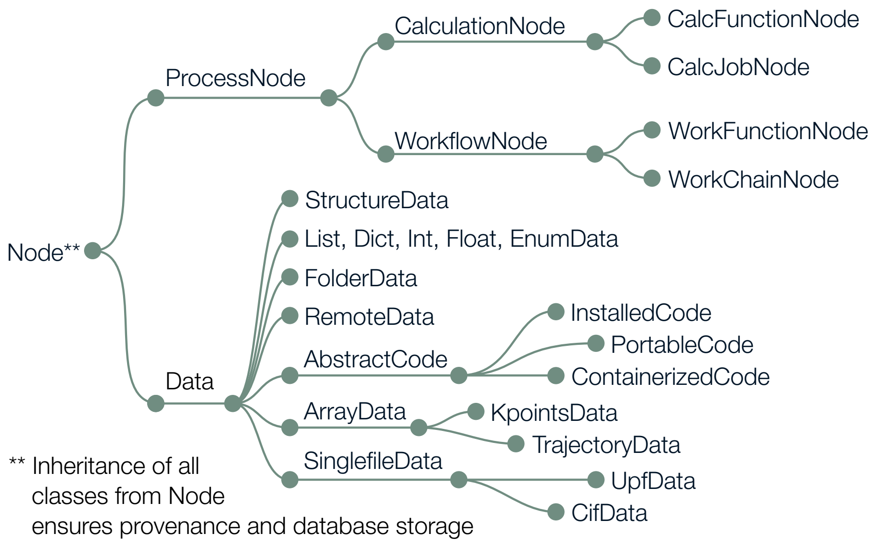
AiiDA to classical file tree:
```
$ verdi process dump <pk>
```

Config options, e.g. caching:
```
$ verdi config list
$ verdi config set \
    caching.default_enabled true
```

Fix what went astray:
```
$ verdi daemon stop
$ verdi process repair
$ verdi daemon start
```

Share/backup your data:
```
$ verdi archive create <archive.aiida> \
    --groups/--nodes <groups/nodes>
$ verdi archive import <archive.aiida>
$ verdi storage backup <backup-path>
```

The AiiDA Node subclasses



** Inheritance of all classes from Node ensures provenance and database storage

Additional web resources (click me)

`aiidalab` `aiida-project` `aiida-shell` `aiida-resource-registry`

`aiida-tutorials` `aiida-submission-controller` `aiida-plugin-cutter`

AiiDA Python imports

ORM, nodes, and Factories

Import aiida-core Node classes from aiida.orm:
```
from aiida.orm import Dict, CalcJobNode
```

Load Nodes via pk, UUID, or label:
```
from aiida.orm import load_node
my_node = load_node(<identifier>)
```

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

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

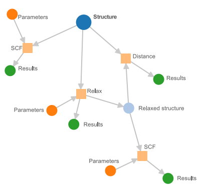
Import CalcJob classes via the CalculationFactory:
```
my_calcjob = CalculationFactory(
    "quantumespresso.pw"
)
```

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 cheat sheet



Main attributes and methods***

Node properties and operations

`label`	Short label
`description`	Verbose description
`pk`	Node ID
`uuid`	Unique ID
`ctime`	Creation time
`mtime`	Modification time
`node_type`	Node type
`store()`	Store node in db

CalcJobNode

`inputs`	CalcJob inputs
`outputs`	CalcJob outputs
`inputs.code`	Executed Code
`computer`	Execution Computer
`get_remote_\ workdir()`	Remote directory
`get_options()`	CalcJob options
`res`	Get `ResultManager`
`res.get_results()`	Results as dict

StructureData

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

Accessed via `node.base.`

`attributes`	Get `NodeAttributes`
`attributes.all`	Attributes as dict
`attributes.get()`	Get specific attribute
`attributes.set()`	Set specific attribute
`extras`	→ Like the `attributes`
`repository`	Get `NodeRepository`
`links`	Get the `NodeLinks`

WorkChain

`spec`	WorkChain specification
`spec.inputs`	Inputs
`spec.outputs`	Outputs
`spec.outline`	Outline of steps
`spec.exit_code`	Exit codes
`ctx`	Context → Data container of WorkChain
`to_context`	Add data to the context

ProcessNode

`exit_status`	Process exit status
`caller`	Parent process that called this process
`called`	Directly called child processes
`is_<property>`	finished / finished_ok / failed / stored / ...
`process_<property>`	class / label / state / status / type
`get_builder_restart()`	Get a prepopulated builder for restarting

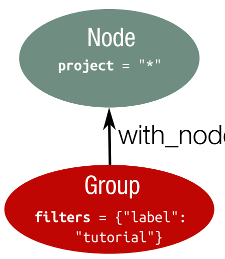
KpointsData

`set_kpoints(<k)`	Set explicit list of kpts
`get_kpoints()`	Get explicit list of kpts
`reciprocal_cell`	Get the reciprocal cell

*** Plus usual property getters/setters
→ but, immutable once stored in db

The QueryBuilder

Fetch all nodes of group "tutorial"

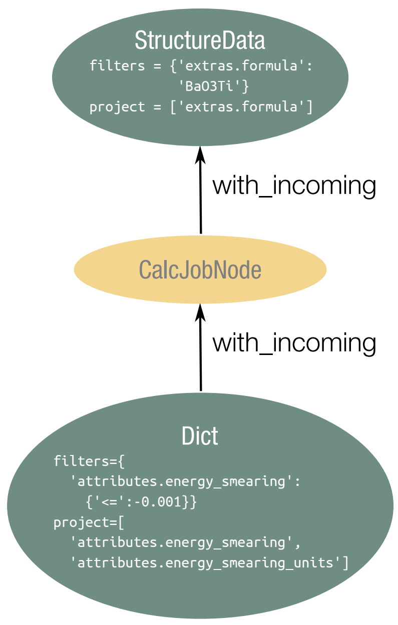


```
from aiida.orm import QueryBuilder

qb = QueryBuilder()
qb.append(Node,
          tag="nodes",
          project="*"
)
qb.append(
    Group,
    with_node="nodes",
    filters={"label": "tutorial"}
)
qb.all()
```

Materials Science example → Smearing energy for BaO_3Ti if smaller than 10^{-4} eV

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


StructureData
filters = {'extras.formula':
'BaO3Ti'}
project = ['extras.formula']
with_incoming
CalcJobNode
with_incoming
Dict
filters={
'attributes.energy_smearing':
{'<=':-0.001}}
project=[
'attributes.energy_smearing',
'attributes.energy_smearing_units']

EPFL swissuniversities INTERSECT