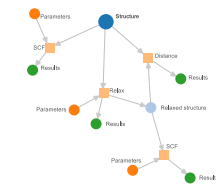
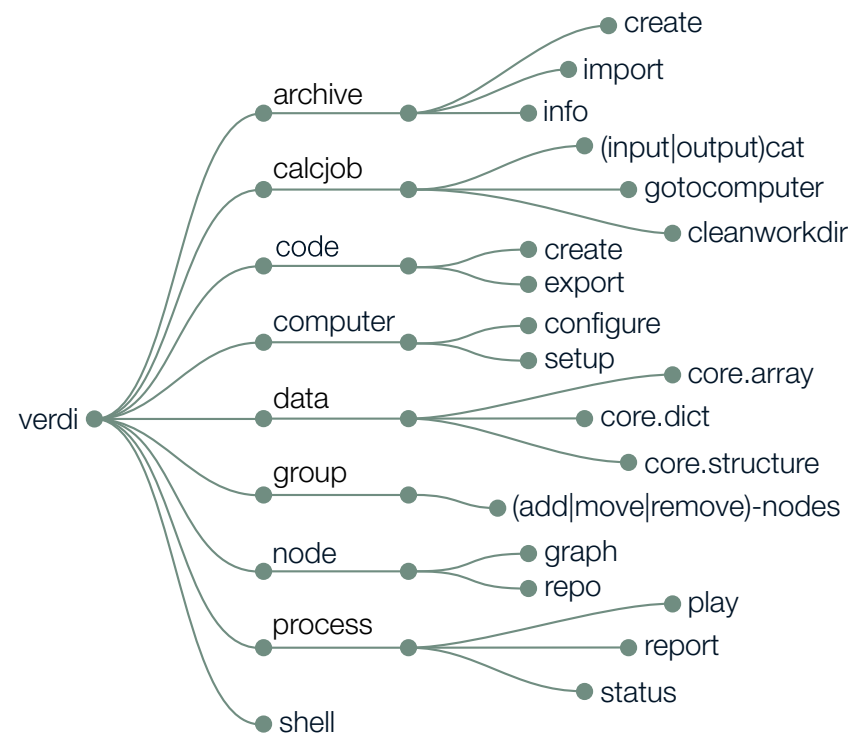


The AiiDA cheat sheet



The verdi command-line API*



*Not exhaustive

*Most options also implement show/list/delete

Tools of the trade

Other verdi tips and tricks

Know what's there:
\$ verdi profile list
\$ verdi user list
\$ verdi plugin list aida.calculations
\$ verdi plugin list aida.workflows

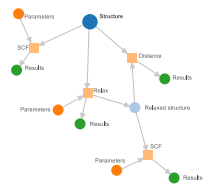
AiiDA to classical file tree:
\$ verdi process dump <pk>
\$ verdi calcjob dump <pk>

Config options, e.g. caching:
\$ verdi config list
\$ verdi config set \
 caching.default_enabled true
\$ verdi config set caching.enabled_for \
 aida.calculations:quantumespresso.pw

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

Share your data:
\$ verdi archive create <archive.aida> \
 --groups/--nodes <groups/nodes>
\$ verdi archive import <archive.aida>

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

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

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

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 / etc.?
process_<property>	class / label / state / status / type
get_builder_restart()	Get a prepopulated builder for restarting

StructureData

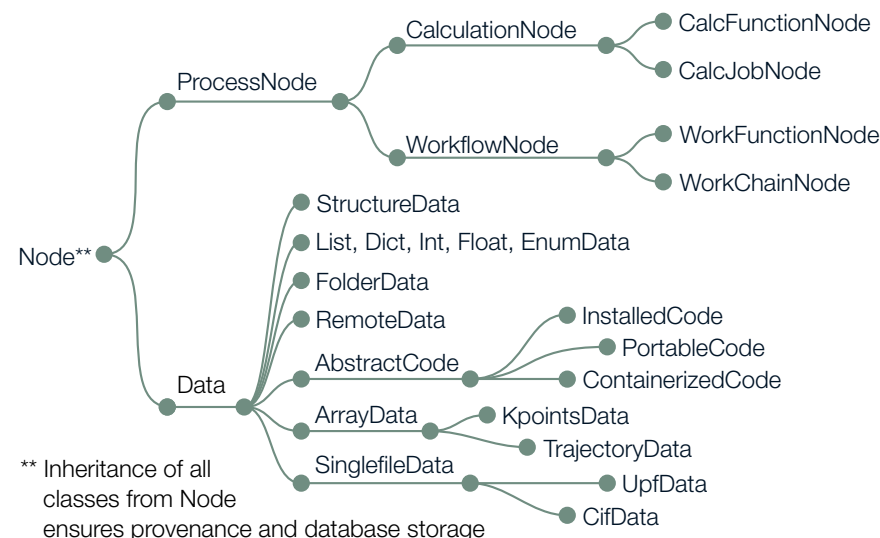
cell	Lattice vectors
get_cell()	Get lattice vectors
set_cell(<c>)	Set lattice vectors
get_cell_volume()	Compute cell volume
pbc	Periodic bound. cond. along each axis
sites	Atomic sites
kinds	Species with masses, symbols, ...
get_formula()	Chemical formula
set_ase(<a>)	Create from ASE
set_pymatgen(<p>)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

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, immutability once stored in db

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"
)

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 BaO₃Ti if smaller than 10⁻⁴ eV

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

