AiiDA-JuTools documentation

Release 0.1.0-dev1

The JuDFT team.

May 26, 2021

Contents

CHAPTER 1

Welcome to documentation of aiida-jutools

The plugin is available at https://github.com/JuDFTteam/aiida-jutools

1.1 Requirements

- ...

1.1.1 User's guide

1.1.1.1 User's guide

1.1.2 Modules provided with aiida-jutools (API reference)

1.1.2.1 Modules provided with aiida-jutools (API reference)

Periodic table of elements; contains: atomic number, period, group and IUPAC index for ordering

Collection of terminal colors

Tools for working with aiida Group entities.

aiida_jutools.util_group.**verdi_group_list**(*projection: List[str] = ['label', 'id', 'type_string'], with_header: bool = True, label_filter: str = None*) → list

Equivalent to CLI "verdi group list -a" (minus user mail address).

Parameters

- **projection** – query projection
- **with_header** – True: first list in return argument is the projection argument
- **label_filter** – optional: only include groups with this substring in their label

Returns list of lists, one entry per projection value, for each group

aiida_jutools.util_group.**move_nodes**(*origin: aiida.orm.groups.Group*, *destination: aiida.orm.groups.Group*)

Move all nodes from one group to another, possibly sub/supergroup.

Parameters

- **origin** – origin group
- **destination** – destination group

Note: if the new group does not exit yet, prefer relabling the group with group.label = new_label.

aiida_jutools.util_group.**get_nodes**(*group_label: str*)

Get all nodes from given group (or subgroup) by label (path).

Deprecated: just use group.nodes, or list(group.nodes).

Parameters **group_label** – e.g. for a subgroup, "groupA/subgroupB/subgroupC".

Returns nodes as generator for efficient iteration (convert via list() to list)

aiida_jutools.util_group.**group_new_nodes**(*new_group_label: str, blacklist: List[aiida.orm.nodes.node.Node] = [<class 'aiida.orm.nodes.data.code.Code'>, <class 'aiida.orm.computers.Computer'>], right_date: datetime.datetime = None, left_date: datetime.datetime = None*)

Groups new nodes with ctime in timerange (left_date,right_date] into new group

If you're working on one project at a time, everytime you finish a project you can use this function to group your nodes. I.e. this is a utility function for a time-linear sequential grouping strategy. Letting the function find the appropriate time range is the standard / recommended usage. If the group already exists and the intended nodes are already added, repeated calls will change nothing.

Parameters

- **new_group_label** – label of new group/subgroup
- **blacklist** – nodes in timerange to exclude from grouping. Normally Code, Computer.
- **right_date** – if not given (usually as datetime.now()), will take right_date = newest ctime, > left_date, of any node, ungrouped nodes included.
- **left_date** – if not given, will take left_date=newest ctime of any grouped node

Returns the new populated, stored, group, or None if no new nodes found

Return type Group or None

aiida_jutools.util_group.**delete_groups**(*group_labels: List[str], skip_nonempty_groups: bool = True, silent: bool = False*)

Delete group(s). Does not delete nodes in group(s). Use delete_groups_with_nodes() for that.

Parameters

- **group_labels** – list of group labels
- **skip_nonempty_groups** – True: skip them. False: don't skip. Nodes get removed from group, not deleted.
- **silent** – True: do not print information.

aiida_jutools.util_group.**delete_groups_with_nodes**(*group_labels: List[str], dry_run: bool = True, verbosity: int = 20, leave_groups: bool = False*)

Delete all nodes in each group (including repo files), then delete the groups themselves.

Parameters

- **group_labels** – list of group labels
- **dry_run** – perform test run. if output looks good, set to false and repeat.
- **verbosity** – 20 = logging.INFO (show node count) (default), 10 = DEBUG (show all uuids), all other: silent.
- **leave_groups** – True: Leave empty groups as is after deleting all nodes in them.

Tools for working with aiida Node objects.

aiida_jutools.util_node.**intersection**(*nodes: List[aiida.orm.nodes.node.Node], others: List[aiida.orm.nodes.node.Node]*)

Computes intersection set of nodes from both lists.

DEVNOTE: outer loop over longer list seems to guarantee symmetry. (without it, computing difference list(set(longer)-set(intersection))==shorter seems to not be guaranteed.)

Parameters

- **nodes** –
- **others** –

Returns

aiida_jutools.util_node.**is_same_node**(*node: aiida.orm.nodes.node.Node, other: aiida.orm.nodes.node.Node, comparator: str = 'uuid'*)

Basic node comparator.

Note: since aiida-core v.1.6.0, the base Node class now evaluates equality based on the node's UUID. Yet, class specific, equality relationships will still override the base class behaviour, for example: Int(99) == Int(99). In case of doubt, prefer this method.

References:

- v1.6.0 https://github.com/aiidateam/aiida-core/blob/develop/CHANGELOG.md#v160—2021-03-15
- v1.5.2- https://www.nature.com/articles/s41597-020-00638-4

Parameters

- **node** – a node.
- **other** – another node.
- **comparator** – "uuid" (default), "pk", or "hash" (warning: slow)

Returns True if same node, False otherwise.

Return type bool

aiida_jutools.util_node.**list_differences**(*calculation_sequence: list, node_type: aiida.orm.nodes.node.Node, member_name: str, outgoing: bool = True*)

Print attributes (e.g. output files) of nodes in list, and only differences in list between subsequent nodes.

Note for comparing Dict nodes, prefer library DeepDiff.

DEVNOTE: TODO redo as tree algorithm navigating via *get_incoming(KkrCalculation)* / *get_outgoing(RemoteData)* given a sinlge node instead of via node list.

Parameters

- **calculation_sequence** –
- **node_type** –
- **member_name** –
- **outgoing** –

Example

```
>>> from aiida.orm import load_node, Dict, FolderData, RemoteData
>>> voro_calc = load_node(1)
>>> kkr_calc = load_node(2)
>>> kkr_calc_converged = load_node(3)
>>> hostGF_calc = load_node(4)
>>> calcs = [voro_calc, kkr_calc, kkr_calc_converged, hostGF_calc]
>>> list_differences(calcs, RemoteData, "listdir")           # outputs.remote_
↪folder
>>> list_differences(calcs, FolderData, "list_object_names") # outputs.retrieved
>>> list_differences(calcs, Dict, "attributes")              # outputs.output_
↪parameters
```

aiida_jutools.util_node.**print_attributes**(*obj*, *obj_name*, *attr_str_list*)

easily print-inspect the values of an aiida object we created.

Parameters

- **obj** – aiida object
- **obj_name** – name
- **attr_str_list** – attributes

Example

```
>>> from aiida.orm import StructureData
>>> from aiida.plugins import DataFactory
>>> StructureData = DataFactory('structure')
>>> # fill in values for copper...
>>> Cu29 = StructureData()
>>> print_attributes(Cu29, "Cu", attribute_string_lists["StructureData"])
```

Tools for working with aiida Code nodes.

aiida_jutools.util_code.**get_code**(*computer_name_pattern: str = ''*, *code_name_pattern: str = ''*, *queue_name: str = ''*)

Find a matching code. If queue_name given, choose code with appropriate architecture.

All arguments are optional. defaults (empty strings), function will query all codes and choose first found. Just try it out with different argument combinations to get a feel for the behavior.

If queue_name given, and applicable for this computer, this will choose the appropriate code under the assumption that different queues (partitions) of the respective computer require the code to be compiled with different architecture. For this to work, it is assumed that the code labels either have a substring which specifies the which specifies the computer queue name, or a substring which specifies the architecture.

All performed substring matches are case-insensitive.

Queue_name <-> architecture code matching available for these computers: - 'iffslurm': FZJ PGI-1 iffslurm cluster.

Queue_name <-> architecture code matching available for these architectures: - 'intel' - 'AMD'

Parameters

- **computer_name_pattern** – substring matching some computer label(s)
- **queue_name** – exact name of the computer queue (slurm: partition)
- **code_name_pattern** – substring matching some code label(s)

Returns closest matching code. if found several, return first warn, but print all matches

Return type Code

Tools for working with aiida Data nodes.

aiida_jutools.util_data.**load_or_rescale_structures**(*input_structure_group*, *output_structure_group_label: str*, *scale_factor*, *set_extra: bool = True*, *dry_run: bool = True*, *silent: bool = False*)

Rescale a group of structures and put them in a new or existing group.

Only input structures which do not already have a rescaled output structure in the output structure group will be rescaled.

Parameters

- **input_structure_group** (*Group*) – group with StructureData nodes to rescale. Ignores other nodes in the group.
- **output_structure_group_label** – name of group for rescaled structures. Create if not exist.
- **scale_factor** (*Float*) – scale factor with which to scale the lattice constant of the input structure
- **set_extra** – True: set extra 'scale_factor' : scale_factor.value to structures rescaled in this run.
- **dry_run** – default True: perform a dry run and print what the method *would* do.
- **silent** – True: do not print info messages

Returns output group of rescaled structures

Return type Group

aiida_jutools.util_data.**query_elemental_structure**(*symbol: str*, *group=None*) → list

Query structures for a single chemical element.

Parameters

- **symbol** – chemical element symbo case-senstive, like 'He'
- **group** – optionally, search only within this group

Returns list of results

aiida_jutools.util_data.**query_modified_input_structure**(*modified_structure*, *invariant_kinds: bool = False*) → list

Given a structure modified via a CalcFunction, query its input structure(s).

Parameters

- **modified_structure** (*StructureData*) – structure modified via a single CalcFunction
- **invariant_kinds** – to make query more precise., assume that the 'kinds' attribute has not been modified.

Returns list of input structures, if any.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index