
Pyoko

Release 0.6.2

January 22, 2016

1	pyoko package	3
1.1	Subpackages	3
1.2	Submodules	8
1.3	pyoko.conf module	8
1.4	pyoko.exceptions module	8
1.5	pyoko.fields module	9
1.6	pyoko.listnode module	10
1.7	pyoko.manage module	13
1.8	pyoko.model module	15
1.9	pyoko.modelmeta module	18
1.10	pyoko.node module	19
1.11	pyoko.registry module	20
1.12	Module contents	20
2	Indices and tables	21
	Python Module Index	23

Contents:

pyoko package

1.1 Subpackages

1.1.1 pyoko.db package

Submodules

pyoko.db.queryset module

this module contains a base class for other db access classes

```
class pyoko.db.queryset.QuerySet (**conf)
```

Bases: object

QuerySet is a lazy data access layer for Riak.

__deepcopy__ (*memo=None*)

A deep copy method that doesn't populate caches and shares Riak client and bucket

_clear_bucket ()

only for development purposes

_compile_query ()

Builds SOLR query and stores it into self.compiled_query

_escape_query (*query, escaped=False*)

Escapes query if it's not already escaped.

Parameters

- **query** – Query value.
- **escaped** (*bool*) – expresses if query already escaped or not.

Returns Escaped query value.

_exec_query ()

Executes solr query if it hasn't already executed.

Returns Self.

_get ()

executes solr query if needed then returns first object according to selected ReturnType (defaults to Model)
:return: pyoko.Model or riak.Object or solr document

`_make_model` (*data*, *riak_obj=None*)

Creates a model instance with the given data.

Parameters

- **data** – Model data returned from DB.
- **riak_obj** –

Returns `pyoko.Model` object.

`_parse_query_key` (*key*, *val*)

Strips query modifier from key and call's the appropriate value modifier.

Parameters

- **key** (*str*) – Query key
- **val** – Query value

Returns Parsed query key and value.

`_parse_query_modifier` (*modifier*, *query_value*)

Parses *query_value* according to *query_type*

Parameters

- **modifier** (*str*) – Type of query. Exact, contains, lte etc.
- **query_value** – Value partition of the query.

Returns Parsed *query_value*.

`_process_params` ()

Adds default row size if it's not given in the query. Converts param values into unicode strings.

Returns Processed `self._solr_params` dict.

`_save_model` (*model=None*)

saves the model instance to riak :return:

`_set_bucket` (*type*, *name*)

prepares bucket, sets index name :param str type: bucket type :param str name: bucket name :return:

`count` ()

counts by executing solr query with rows=0 parameter :return: number of objects matches to the query
:rtype: int

`data` ()

set return type as riak objects instead of pyoko models

`distinct_values_of` (*field*)

`exclude` (***filters*)

Applies query filters for excluding matching records from result set.

Parameters ****filters** – Query filters as keyword arguments.

Returns Self. Queryset object.

Examples

```
>>> Person.objects.exclude(age=None)
>>> Person.objects.filter(name__startswith='jo').exclude(age__lte=16)
```


filter (***filters*)

Applies given query filters.

Parameters ****filters** – Query filters as keyword arguments.

Returns Self. Queryset object.

Examples

```
>>> Person.objects.filter(name='John') # same as .filter(name__exact='John')
>>> Person.objects.filter(age__gte=16, name__startswith='jo')
>>> # Assume u1 and u2 as related model instances.
>>> Person.objects.filter(work_unit__in=[u1, u2], name__startswith='jo')
```

get (*key=None, **kwargs*)

Ensures that only one result is returned from DB and raises an exception otherwise. Can work in 3 different way.

- If no argument is given, only does “ensuring about one and only object” job.
- If key given as only argument, retrieves the object from DB.
- if query filters given, implicitly calls filter() method.

Raises `MultipleObjectsReturned` – If there is more than one (1) record is returned.

get_or_create (*defaults=None, **kwargs*)

Looks up an object with the given kwargs, creating a new one if necessary.

Parameters

- **defaults** (*dict*) – Used when we create a new object. Must map to fields of the model.
- ****kwargs** – Used both for filtering and new object creation.

Returns A tuple of (object, created), where created is a boolean variable specifies whether the object was newly created or not.

Example

In the following example, *code* and *name* fields are used to query the DB.

```
obj, is_new = Permission.objects.get_or_create({'description': desc},
                                              code=code, name=name)
```

{description: desc} dict is just for new creations. If we can't find any records by filtering on *code* and *name*, then we create a new object by using all of the inputs.

or_filter (***filters*)

Works like “filter” but joins given filters with OR operator.

Parameters ****filters** – Query filters as keyword arguments.

Returns Self. Queryset object.

Example

```
>>> Person.objects.or_filter(age__gte=16, name__startswith='jo')
```

raw (*query*, ***params*)

make a raw query :param str query: solr query :param dict params: solr parameters

search_on (**fields*, ***query*)

Search for query on given fields.

Query modifier can be one of these: # * exact * contains * startswith * endswith * range * lte * gte

Parameters

- ***fields** (*str*) – Field list to be searched on
- ****query** – Search query. While it's implemented as ****kwargs** we only support one (first) keyword argument.

Returns Self. Queryset object.

Examples

```
>>> Person.objects.search_on('name', 'surname', contains='john')
>>> Person.objects.search_on('name', 'surname', startswith='jo')
```

set_params (***params*)

add/update solr query parameters

solr ()

set return type for raw solr docs

class pyoko.db.queryset.ReturnType

Bases: enum.Enum

Model = <ReturnType.Model: 3>

Object = <ReturnType.Object: 2>

Solr = <ReturnType.Solr: 1>

pyoko.db.connection module

riak client configuration

pyoko.db.schema_update module

class pyoko.db.schema_update.FakeContext

Bases: object

has_permission (*perm*)

class pyoko.db.schema_update.SchemaUpdater (*registry*, *bucket_names*, *threads*, *force*)

Bases: object

traverses trough all models, collects fields marked for index or store in solr then creates a solr schema for these fields.

FIELD_TEMPLATE = '<field type="{type}" name="{name}" indexed="{index}" stored="{store}" multiValued="{multi}"

static apply_schema (*client, force, job_pack*)

riak doesn't support schema/index updates (<http://git.io/vLOTS>)

as a workaround, we create a temporary index, attach it to the bucket, delete the old index/schema, re-create the index with new schema, assign it to bucket, then delete the temporary index.

Parameters

- **new_schema** (*byte*) – compiled schema
- **bucket_name** (*str*) – name of schema, index and bucket.

Returns True or False

Return type bool

compile_schema (*fields*)

joins schema fields with base solr schema

Parameters **fields** (*list[str]*) – field list

Returns compiled schema

Return type byte

create_report ()

creates a text report for the human user :return: str

classmethod get_schema_fields (*fields*)

Parameters **fields** (*list[(,)]*) – field props tuple list

Return type list[str]

Returns schema fields list

run ()

`pyoko.db.schema_update.get_schema_from_solr(index_name)`

`pyoko.db.schema_update.wait_for_schema_creation(index_name)`

`pyoko.db.schema_update.wait_for_schema_deletion(index_name)`

Module contents

1.1.2 pyoko.lib package

Submodules

pyoko.lib.py2map module

tools to convert Python dicts to / from riak Maps

pyoko.lib.utils module

class `pyoko.lib.utils.MyEncoder` (*skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, encoding='utf-8', default=None*)

Bases: `json.encoder.JSONEncoder`

default (*obj*)

```
class pyoko.lib.utils.SimpleChoicesManager
    Bases: object

    static get_all (choices)
```

```
class pyoko.lib.utils.SimpleRiakFileManager
    Bases: object

    store_file (**kw)
```

```
pyoko.lib.utils.add_to_path()
```

```
pyoko.lib.utils.getScriptPath()
```

```
pyoko.lib.utils.get_object_from_path(path)
    Import's object from given Python path.
```

```
pyoko.lib.utils.grayed(*args)
```

```
pyoko.lib.utils.pprnt(input, return_data=False)
    Prettier print for nested data
```

Parameters

- **input** – Input data
- **return_data** (*bool*) – Default False. Print outs if False, returns if True.

Returns None | Pretty formatted text representation of input data.

```
pyoko.lib.utils.random_word(length)
```

```
pyoko.lib.utils.to_camel(s)
```

Parameters **s** (*string*) – under_scored string to be CamelCased

Returns CamelCase version of input

Return type str

```
pyoko.lib.utils.un_camel(input)
```

```
pyoko.lib.utils.un_camel_id(input)
    uncamel for id fields :param input: :return:
```

Module contents

1.2 Submodules

1.3 pyoko.conf module

```
class pyoko.conf.Settings
    Bases: object
```

1.4 pyoko.exceptions module

```
exception pyoko.exceptions.MultipleObjectsReturned
    Bases: pyoko.exceptions.PyokoError
```

The query returned multiple objects when only one was expected.

exception `pyoko.exceptions.NoSuchObjectError`

Bases: `pyoko.exceptions.PyokoError`

exception `pyoko.exceptions.NotCompatible`

Bases: `pyoko.exceptions.PyokoError`

Incorrect usage of method / function

exception `pyoko.exceptions.ObjectDoesNotExist`

Bases: `pyoko.exceptions.PyokoError`

exception `pyoko.exceptions.PyokoError`

Bases: `exceptions.Exception`

exception `pyoko.exceptions.ValidationError`

Bases: `pyoko.exceptions.PyokoError`

1.5 pyoko.fields module

class `pyoko.fields.BaseField` (`title='', default=None, required=True, index=False, type=None, store=False, choices=None, order=None, **kwargs`)

Bases: `object`

`__load_data` (`instance, value`)

for some field types (eg:date, datetime) we treat differently to data that came from db and given by user

`clean_value` (`val`)

`creation_counter` = 2

`default_value` = None

`validate` (`val`)

class `pyoko.fields.Boolean` (`title='', default=None, required=True, index=False, type=None, store=False, choices=None, order=None, **kwargs`)

Bases: `pyoko.fields.BaseField`

`clean_value` (`val`)

`solr_type` = 'boolean'

class `pyoko.fields.Date` (`*args, **kwargs`)

Bases: `pyoko.fields.BaseField`

`clean_value` (`val`)

`solr_type` = 'date'

class `pyoko.fields.DateTimeField` (`*args, **kwargs`)

Bases: `pyoko.fields.BaseField`

`clean_value` (`val`)

`solr_type` = 'date'

class `pyoko.fields.File` (`*args, **kwargs`)

Bases: `pyoko.fields.BaseField`

`clean_value` (`val`)

`val` = :param dict `val`: {"content":"","name":"","ext":"","type":""} :return:

`file_manager`

```
    solr_type = 'file'

class pyoko.fields.Float(title='', default=None, required=True, index=False, type=None,
                        store=False, choices=None, order=None, **kwargs)
    Bases: pyoko.fields.BaseField
    Numeric field that holds float data.

    clean_value(val)

    solr_type = 'float'

class pyoko.fields.Id(*arg, **kwargs)
    Bases: pyoko.fields.BaseField

    clean_value(val)

    solr_type = 'string'

class pyoko.fields.Integer(title='', default=None, required=True, index=False, type=None,
                          store=False, choices=None, order=None, **kwargs)
    Bases: pyoko.fields.BaseField

    clean_value(val)

    default_value = 0

    solr_type = 'int'

class pyoko.fields.String(title='', default=None, required=True, index=False, type=None,
                         store=False, choices=None, order=None, **kwargs)
    Bases: pyoko.fields.BaseField

    solr_type = 'string'

class pyoko.fields.Text(title='', default=None, required=True, index=False, type=None, store=False,
                      choices=None, order=None, **kwargs)
    Bases: pyoko.fields.BaseField
    Text field.

    solr_type = 'text_general'

class pyoko.fields.TimeStamp(*args, **kwargs)
    Bases: pyoko.fields.BaseField

    clean_value(val)

    solr_type = 'long'
```

1.6 pyoko.listnode module

This module holds the ListNode implementation of Pyoko Models.

ListNode's are used to model ManyToMany relations and other list like data types on a Model.

```
class pyoko.listnode.ListNode(**kwargs)
    Bases: pyoko.node.Node
```

ListNode's are used to store list of field sets. Their DB representation look like list of dicts:

```
class Student(Model):
    class Lectures(ListNode):
        name = field.String()
```

```

        code = field.String(required=False)

    st = Student()
    st.Lectures(name="Math101", code='M1')
    st.Lectures(name="Math102", code='M2')
    st.clean_value()
    {
        'deleted': False,
        'timestamp': None
        'lectures': [
            {'code': 'M1', 'name': 'Math101'},
            {'code': 'M2', 'name': 'Math102'},
        ]
    }

```

Notes

- Currently we disregard the ordering of ListNode items.

__call__ (***kwargs*)

Stores created instance in node_stack and returns it's reference to callee

__delattr__

x.__delattr__('name') <==> del x.name

__delitem__ (*obj*)

Allow usage of “del” statement on ListNodes with bracket notation.

Parameters *obj* – ListNode item or relation key.

Raises *TypeError* – If it's called on a ListNode item (instead of ListNode's itself)

__format__ ()

default object formatter

__getattr__

x.__getattr__('name') <==> x.name

__hash__

__reduce__ ()

helper for pickle

__reduce_ex__ ()

helper for pickle

__repr__ ()

This works for two different object:

- Main ListNode object
- Items of the ListNode (like instance of a class) which created while iterating on main ListNode object

Returns String representation of object.

__setattr__

x.__setattr__('name', value) <==> x.name = value

__sizeof__ () → int

size of object in memory, in bytes

__generate_instances ()

ListNode item generator. Will be used internally by `__iter__` and `__getitem__`

Yields ListNode items (instances)

__get_linked_model_key ()

Only one linked model can represent a listnode instance,

Returns The first linked models key if exists otherwise None

__load_data (*data*, *from_db=False*)

Stores the data at self._data, actual object creation done at `__generate_instances()`

Parameters

- **data** (*list*) – List of dicts.
- **from_db** (*bool*) – Default False. Is this data coming from DB or not.

__make_instance (*node_data*)

Create a ListNode instance from node_data

Parameters **node_data** (*dict*) – Data to create ListNode item.

Returns ListNode item.

add (***kwargs*)

Stores node data without creating an instance of it. This is more efficient if node instance is not required.

Parameters **kwargs** – attributes of the ListNode

clean_value ()

Populates json serialization ready data. This is the method used to serialize and store the object data in to DB

Returns List of dicts.

clear ()

Clear outs the list node.

Raises `TypeError` – If it's called on a ListNode item (instead of ListNode's itself)

get_field (*field_name*)

get_humane_value (*name*)

Returns a human readable/meaningful value for the field

Parameters **name** (*str*) – Model field name

Returns Human readable field value

get_link (***kw*)

get_links (***kw*)

get_verbose_name ()

remove ()

Removes an item from ListNode.

Raises `TypeError` – If it's called on a ListNode item (instead of ListNode's itself)

Note: Parent object should be explicitly saved.

1.7 pyoko.manage module

command line management interface

class `pyoko.manage.Command` (*manager*)

Bases: `object`

Command object is a thin wrapper around Python's powerful `argparse` module. Holds the given command line parameters in `self.manager.args`

CMD_NAME

name of your command

HELP

help texts starts with "Rl" will be parsed as raw text

PARAMS

A dictionary list with following possible values.

- name: name of parameter
- help: help text for parameter. Parsed as raw if starts with "Rl"
- required: Optional. Set True if this is a required parameter.
- default: Optional. Define a default value for the parameter
- action: 'store_true' see the official `argparse` documentation for more info

run()

This is where the things are done. You should override this method in your command class.

class `pyoko.manage.CommandRegistry` (*mcs, name, bases, attrs*)

Bases: `type`

classmethod `add_command` (*command_model*)

classmethod `get_commands` ()

`CommandRegistry.registry` [{}]

class `pyoko.manage.DumpData` (*manager*)

Bases: `pyoko.manage.Command`

CHOICES = ('csv', 'json', 'json_tree', 'pretty')

CMD_NAME = 'dump_data'

CSV = 'csv'

HELP = 'Dumps all data to stdout or to given file'

JSON = 'json'

DumpData.PARAMS [{}]

PRETTY = 'pretty'

TREE = 'json_tree'

run()

class `pyoko.manage.FindDuplicateKeys` (*manager*)

Bases: `pyoko.manage.Command`

CMD_NAME = '_find_dups'

```
    HELP = 'finds duplicate keys, to help debugging'

    run ()

class pyoko.manage.FlushDB (manager)
    Bases: pyoko.manage.Command

    CMD_NAME = 'flush_model'

    HELP = 'REALLY DELETES the contents of buckets'

    FlushDB.PARAMS [{}]

    run ()

class pyoko.manage.LoadData (manager)
    Bases: pyoko.manage.Command

    Loads previously dumped data into DB.

    CHOICES = ('csv', 'json', 'json_tree', 'pretty')

    CMD_NAME = 'load_data'

    CSV = 'csv'

    HELP = 'Reads JSON data from given file and populates models'

    JSON = 'json'

    LoadData.PARAMS [{}]

    PRETTY = 'pretty'

    TREE = 'json_tree'

    prepare_buckets ()
        loads buckets to bucket cache. disables the default json encoders if CSV is selected

    read_file (file_path)

    read_json_per_line (file)

    read_per_line (file)

    read_whole_file (file)

    run ()

    save_obj (bucket_name, key, val)

class pyoko.manage.ManagementCommands (args=None)
    Bases: object

    All CLI commands executed by this class. You can create your own commands by extending Command class

    parse_args (args)

class pyoko.manage.SchemaUpdate (manager)
    Bases: pyoko.manage.Command

    CMD_NAME = 'migrate'

    HELP = 'Creates/Updates SOLR schemas for given model(s)'

    SchemaUpdate.PARAMS [{}]

    run ()
```

```

class pyoko.manage.Shell(manager)
    Bases: pyoko.manage.Command

    CMD_NAME = 'shell'

    HELP = 'Run IPython shell'

    Shell.PARAMS [{}]

    run()

class pyoko.manage.SmartFormatter(prog, indent_increment=2, max_help_position=24,
    width=None)
    Bases: argparse.HelpFormatter

class pyoko.manage.TestGetKeys(manager)
    Bases: pyoko.manage.Command

    CMD_NAME = '_test_get_keys'

    HELP = 'tests the correctness of the bucket.get_keys()'

    run()

```

1.8 pyoko.model module

```

class pyoko.model.LinkProxy(link_to, one_to_one=False, verbose_name=None, re-
    verse_name=None)
    Bases: object

    Proxy object for "self" referencing model relations .. rubric:: Example

```

```

class Unit(Model):
    name = field.String("Name")
    parent = LinkProxy('Unit', verbose_name='Upper unit', reverse_name='sub_units')

```

```

__delattr__
    x.__delattr__('name') <==> del x.name

__format__()
    default object formatter

__getattr__
    x.__getattr__('name') <==> x.name

__hash__

__reduce__()
    helper for pickle

__reduce_ex__()
    helper for pickle

__repr__

__setattr__
    x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
    size of object in memory, in bytes

__str__

```

```
class pyoko.model.Model (context=None, **kwargs)
    Bases: pyoko.node.Node
```

This is base class for any model object.

Field instances are used as model attributes to represent values.

```
class Permission(Model):
    name = field.String("Name")
    code = field.String("Code Name")

    def __unicode__(self):
        return "%s %s" % (self.name, self.code)
```

Models may have inner classes to represent ManyToMany relations, inner data nodes or lists.

```
__delattr__
    x.__delattr__('name') <==> del x.name
```

```
__eq__ (other)
    Equivalence of two model instance depends on uniformity of their self._data and self.key.
```

```
__format__ ()
    default object formatter
```

```
__getattr__
    x.__getattr__('name') <==> x.name
```

```
__reduce__ ()
    helper for pickle
```

```
__reduce_ex__ ()
    helper for pickle
```

```
__setattr__
    x.__setattr__('name', value) <==> x.name = value
```

```
__sizeof__ () → int
    size of object in memory, in bytes
```

```
__apply_cell_filters (context)
```

Applies the field restrictions based on the return value of the context's "has_permission()" method.
Stores them on self._unpermitted_fields.

Returns List of unpermitted fields names.

```
__handle_changed_fields (old_data)
    Looks for changed relation fields between new and old data (before/after save). Creates back_link refer-
    ences for updated fields.
```

Parameters `old_data` – Object's data before save.

```
__update_new_linked_model (linked_mdl_ins, link)
    Iterates through linked_models of given model instance to match it's "reverse" with given link's "field"
    values.
```

```
clean_value ()
    generates a json serializable representation of the model data :rtype: dict :return: riak ready python dict
```

```
delete ()
    This method just flags the object as "deleted" and saves it to DB.
```

exist

Used to check if a relation is exist or a model instance is saved to DB or not.

Returns True if this model instance stored in DB and has a key and False otherwise.

Examples

```
>>> class Student (Model) :
>>>     #...
>>>     adviser = Person()
>>>
>>> if student.adviser.exist:
>>>     # do something
```

get_choices_for (*field*)

Get the choices for the given fields.

Parameters **field** (*str*) – Name of field.

Returns List of tuples. [(name, value),...]

get_field (*field_name*)**get_humane_value** (*name*)

Returns a human readable/meaningful value for the field

Parameters **name** (*str*) – Model field name

Returns Human readable field value

get_link (***kw*)**get_links** (***kw*)**get_unpermitted_fields** ()

Gives unpermitted fields for current context/user.

Returns List of unpermitted field names.

get_verbose_name ()

Returns Verbose name of the model instance

is_in_db ()

Deprecated: Use “exist” property instead.

objects

alias of QuerySet

post_save ()

Called after object save. Can be overridden to do things that should be done after object saved to DB.

pre_save ()

Called before object save. Can be overridden to do things that should be done just before object saved to DB.

prnt ()

Prints DB data representation of the object.

static row_level_access (*context, objects*)

If defined, will be called just before query compiling step and it's output summed up to existing query filter.

Can be used to implement context-aware implicit filtering. You can define your query filters in here to enforce row level access control.

Parameters

- **context** – An object that contain required user attributes and permissions.
- **objects** (*Queryset*) – QuerySet object.

Examples

```
>>> return objects.filter(user=context.user)
```

save (*internal=False*)
Save's object to DB.

Do not override this method, use `pre_save` and `post_save` methods.

Parameters **internal** (*bool*) – True if called within model. Used to prevent unnecessary calls to `pre_save` and `post_save` methods.

Returns Saved model instance.

set_data (*data, from_db=False*)
Fills the object's fields with given data dict. Internally calls the `self._load_data()` method.

Parameters

- **data** (*dict*) – Data to fill object's fields.
- **from_db** (*bool*) – if data coming from db then we will
- **related field type's _load_data method** (*use*) –

Returns Self. Returns objects itself for chainability.

1.9 pyoko.modelmeta module

class `pyoko.modelmeta.ModelMeta` (*mcs, name, bases, attrs*)
Bases: `type`

Metaclass that process model classes.

static process_attributes_of_node (*attrs, node_name, class_type*)
prepare the model fields, nodes and relations

Parameters

- **node_name** (*str*) – name of the node we are currently processing
- **attrs** (*dict*) – attribute dict
- **class_type** (*str*) – Type of class. Can be one of these: 'ListNode', 'Model', 'Node'

static process_models (*attrs, base_model_class*)
Attach default fields and meta options to models

static process_objects (*kls*)
Applies default Meta properties.

1.10 pyoko.node module

class `pyoko.node.FakeContext`

Bases: `object`

this fake context object can be used to use ACL limited models from shell

has_permission (*perm*)

class `pyoko.node.LazyModel` (*wrapped, verbose_name*)

Bases: `Proxy`

exist

get_verbose_name ()

key = `None`

verbose_name = `None`

class `pyoko.node.Node` (***kwargs*)

Bases: `object`

We store node classes in `_nodes[]` attribute at `ModelMeta`, then replace them with their instances at `_instantiate_nodes()`

Likewise we store linked models in `_linked_models[]`

Since fields are defined as descriptors, they can access to instance they called from but to access their methods and attributes, we're copying fields themselves into `self._fields[]` attribute. So, we get values of fields from `self._field_values[]` and access to fields themselves from `self._fields[]`

_collect_index_fields (*in_multi=False*)

Collects fields which will be indexed.

Parameters *in_multi* (*bool*) – if we are in a `ListNode` or not

Returns [(*field_name*, *solr_type*, *is_indexed*, *is_stored*, *is_multi*)]

_instantiate_nodes ()

instantiate all nodes

_load_data (*data, from_db=False*)

With the data returned from riak: - fills model's fields, nodes and listnodes - instantiates linked model instances

Parameters

- **from_db** (*bool*) – if data coming from db instead of calling `self._set_fields_values()` we simply use field's `_load_data` method.
- **data** (*dict*) –

Returns `self`

_path_of (*prop*)

returns the dotted path of the given model attribute

_set_fields_values (*kwargs*)

Fill the fields of this node

Parameters *kwargs* – Field values

clean_value ()

generates a json serializable representation of the model data :rtype: dict :return: riak ready python dict

```
classmethod get_field (field_name)
get_humane_value (name)
    Returns a human readable/meaningful value for the field

    Parameters name (str) – Model field name

    Returns Human readable field value

classmethod get_link ( **kw )
classmethod get_links ( **kw )
get_verbose_name ( )
```

1.11 pyoko.registry module

```
class pyoko.registry.Registry
    Bases: object

    get_apps ( )
    get_base_models ( )
    get_model (model_name)
    get_models_by_apps ( )
    get_models_of_app (app_name)
    register_model (mdl)
```

1.12 Module contents

Indices and tables

- `genindex`
- `search`

p

- [pyoko](#), 20
- [pyoko.conf](#), 8
- [pyoko.db](#), 7
 - [pyoko.db.connection](#), 6
 - [pyoko.db.queryset](#), 3
 - [pyoko.db.schema_update](#), 6
- [pyoko.exceptions](#), 8
- [pyoko.fields](#), 9
- [pyoko.lib](#), 8
 - [pyoko.lib.py2map](#), 7
 - [pyoko.lib.utils](#), 7
- [pyoko.listnode](#), 10
- [pyoko.manage](#), 13
- [pyoko.model](#), 15
- [pyoko.modelmeta](#), 18
- [pyoko.node](#), 19
- [pyoko.registry](#), 20

Symbols

- `__call__()` (pyoko.listnode.ListNode method), 11
 - `__deepcopy__()` (pyoko.db.queryset.QuerySet method), 3
 - `__delattr__` (pyoko.listnode.ListNode attribute), 11
 - `__delattr__` (pyoko.model.LinkProxy attribute), 15
 - `__delattr__` (pyoko.model.Model attribute), 16
 - `__delitem__()` (pyoko.listnode.ListNode method), 11
 - `__eq__()` (pyoko.model.Model method), 16
 - `__format__()` (pyoko.listnode.ListNode method), 11
 - `__format__()` (pyoko.model.LinkProxy method), 15
 - `__format__()` (pyoko.model.Model method), 16
 - `__getattr__` (pyoko.listnode.ListNode attribute), 11
 - `__getattr__` (pyoko.model.LinkProxy attribute), 15
 - `__getattr__` (pyoko.model.Model attribute), 16
 - `__hash__` (pyoko.listnode.ListNode attribute), 11
 - `__hash__` (pyoko.model.LinkProxy attribute), 15
 - `__reduce__()` (pyoko.listnode.ListNode method), 11
 - `__reduce__()` (pyoko.model.LinkProxy method), 15
 - `__reduce__()` (pyoko.model.Model method), 16
 - `__reduce_ex__()` (pyoko.listnode.ListNode method), 11
 - `__reduce_ex__()` (pyoko.model.LinkProxy method), 15
 - `__reduce_ex__()` (pyoko.model.Model method), 16
 - `__repr__` (pyoko.model.LinkProxy attribute), 15
 - `__repr__()` (pyoko.listnode.ListNode method), 11
 - `__setattr__` (pyoko.listnode.ListNode attribute), 11
 - `__setattr__` (pyoko.model.LinkProxy attribute), 15
 - `__setattr__` (pyoko.model.Model attribute), 16
 - `__sizeof__()` (pyoko.listnode.ListNode method), 11
 - `__sizeof__()` (pyoko.model.LinkProxy method), 15
 - `__sizeof__()` (pyoko.model.Model method), 16
 - `__str__` (pyoko.model.LinkProxy attribute), 15
 - `__apply_cell_filters()` (pyoko.model.Model method), 16
 - `__clear_bucket()` (pyoko.db.queryset.QuerySet method), 3
 - `__collect_index_fields()` (pyoko.node.Node method), 19
 - `__compile_query()` (pyoko.db.queryset.QuerySet method), 3
 - `__escape_query()` (pyoko.db.queryset.QuerySet method), 3
 - `__exec_query()` (pyoko.db.queryset.QuerySet method), 3
 - `__generate_instances()` (pyoko.listnode.ListNode method), 12
 - `__get()` (pyoko.db.queryset.QuerySet method), 3
 - `__get_linked_model_key()` (pyoko.listnode.ListNode method), 12
 - `__handle_changed_fields()` (pyoko.model.Model method), 16
 - `__instantiate_nodes()` (pyoko.node.Node method), 19
 - `__load_data()` (pyoko.fields.BaseField method), 9
 - `__load_data()` (pyoko.listnode.ListNode method), 12
 - `__load_data()` (pyoko.node.Node method), 19
 - `__make_instance()` (pyoko.listnode.ListNode method), 12
 - `__make_model()` (pyoko.db.queryset.QuerySet method), 3
 - `__parse_query_key()` (pyoko.db.queryset.QuerySet method), 4
 - `__parse_query_modifier()` (pyoko.db.queryset.QuerySet method), 4
 - `__path_of()` (pyoko.node.Node method), 19
 - `__process_params()` (pyoko.db.queryset.QuerySet method), 4
 - `__save_model()` (pyoko.db.queryset.QuerySet method), 4
 - `__set_bucket()` (pyoko.db.queryset.QuerySet method), 4
 - `__set_fields_values()` (pyoko.node.Node method), 19
 - `__update_new_linked_model()` (pyoko.model.Model method), 16
- ## A
- `add()` (pyoko.listnode.ListNode method), 12
 - `add_command()` (pyoko.manage.CommandRegistry class method), 13
 - `add_to_path()` (in module pyoko.lib.utils), 8
 - `apply_schema()` (pyoko.db.schema_update.SchemaUpdater static method), 7
- ## B
- `BaseField` (class in pyoko.fields), 9
 - `Boolean` (class in pyoko.fields), 9
- ## C
- `CHOICES` (pyoko.manage.DumpData attribute), 13
 - `CHOICES` (pyoko.manage.LoadData attribute), 14
 - `clean_value()` (pyoko.fields.BaseField method), 9

`clean_value()` (pyoko.fields.Boolean method), 9
`clean_value()` (pyoko.fields.Date method), 9
`clean_value()` (pyoko.fields.DateTime method), 9
`clean_value()` (pyoko.fields.File method), 9
`clean_value()` (pyoko.fields.Float method), 10
`clean_value()` (pyoko.fields.Id method), 10
`clean_value()` (pyoko.fields.Integer method), 10
`clean_value()` (pyoko.fields.TimeStamp method), 10
`clean_value()` (pyoko.listnode.ListNode method), 12
`clean_value()` (pyoko.model.Model method), 16
`clean_value()` (pyoko.node.Node method), 19
`clear()` (pyoko.listnode.ListNode method), 12
`CMD_NAME` (pyoko.manage.Command attribute), 13
`CMD_NAME` (pyoko.manage.DumpData attribute), 13
`CMD_NAME` (pyoko.manage.FindDuplicateKeys attribute), 13
`CMD_NAME` (pyoko.manage.FlushDB attribute), 14
`CMD_NAME` (pyoko.manage.LoadData attribute), 14
`CMD_NAME` (pyoko.manage.SchemaUpdate attribute), 14
`CMD_NAME` (pyoko.manage.Shell attribute), 15
`CMD_NAME` (pyoko.manage.TestGetKeys attribute), 15
`Command` (class in pyoko.manage), 13
`CommandRegistry` (class in pyoko.manage), 13
`compile_schema()` (pyoko.db.schema_update.SchemaUpdater method), 7
`count()` (pyoko.db.queryset.QuerySet method), 4
`create_report()` (pyoko.db.schema_update.SchemaUpdater method), 7
`creation_counter` (pyoko.fields.BaseField attribute), 9
`CSV` (pyoko.manage.DumpData attribute), 13
`CSV` (pyoko.manage.LoadData attribute), 14

D

`data()` (pyoko.db.queryset.QuerySet method), 4
`Date` (class in pyoko.fields), 9
`DateTime` (class in pyoko.fields), 9
`default()` (pyoko.lib.utils.MyEncoder method), 7
`default_value` (pyoko.fields.BaseField attribute), 9
`default_value` (pyoko.fields.Integer attribute), 10
`delete()` (pyoko.model.Model method), 16
`distinct_values_of()` (pyoko.db.queryset.QuerySet method), 4
`DumpData` (class in pyoko.manage), 13

E

`exclude()` (pyoko.db.queryset.QuerySet method), 4
`exist` (pyoko.model.Model attribute), 16
`exist` (pyoko.node.LazyModel attribute), 19

F

`FakeContext` (class in pyoko.db.schema_update), 6
`FakeContext` (class in pyoko.node), 19

`FIELD_TEMPLATE` (pyoko.db.schema_update.SchemaUpdater attribute), 6

`File` (class in pyoko.fields), 9
`file_manager` (pyoko.fields.File attribute), 9
`filter()` (pyoko.db.queryset.QuerySet method), 4
`FindDuplicateKeys` (class in pyoko.manage), 13
`Float` (class in pyoko.fields), 10
`FlushDB` (class in pyoko.manage), 14

G

`get()` (pyoko.db.queryset.QuerySet method), 5
`get_all()` (pyoko.lib.utils.SimpleChoicesManager static method), 8
`get_apps()` (pyoko.registry.Registry method), 20
`get_base_models()` (pyoko.registry.Registry method), 20
`get_choices_for()` (pyoko.model.Model method), 17
`get_commands()` (pyoko.manage.CommandRegistry class method), 13
`get_field()` (pyoko.listnode.ListNode method), 12
`get_field()` (pyoko.model.Model method), 17
`get_field()` (pyoko.node.Node class method), 19
`get_humane_value()` (pyoko.listnode.ListNode method), 12
`get_humane_value()` (pyoko.model.Model method), 17
`get_humane_value()` (pyoko.node.Node method), 20
`get_link()` (pyoko.listnode.ListNode method), 12
`get_link()` (pyoko.model.Model method), 17
`get_link()` (pyoko.node.Node class method), 20
`get_links()` (pyoko.listnode.ListNode method), 12
`get_links()` (pyoko.model.Model method), 17
`get_links()` (pyoko.node.Node class method), 20
`get_model()` (pyoko.registry.Registry method), 20
`get_models_by_apps()` (pyoko.registry.Registry method), 20
`get_models_of_app()` (pyoko.registry.Registry method), 20
`get_object_from_path()` (in module pyoko.lib.utils), 8
`get_or_create()` (pyoko.db.queryset.QuerySet method), 5
`get_schema_fields()` (pyoko.db.schema_update.SchemaUpdater class method), 7
`get_schema_from_solr()` (in module pyoko.db.schema_update), 7
`get_unpermitted_fields()` (pyoko.model.Model method), 17
`get_verbose_name()` (pyoko.listnode.ListNode method), 12
`get_verbose_name()` (pyoko.model.Model method), 17
`get_verbose_name()` (pyoko.node.LazyModel method), 19
`get_verbose_name()` (pyoko.node.Node method), 20
`getScriptPath()` (in module pyoko.lib.utils), 8
`grayed()` (in module pyoko.lib.utils), 8

H

has_permission() (pyoko.db.schema_update.FakeContext method), 6
 has_permission() (pyoko.node.FakeContext method), 19
 HELP (pyoko.manage.Command attribute), 13
 HELP (pyoko.manage.DumpData attribute), 13
 HELP (pyoko.manage.FindDuplicateKeys attribute), 13
 HELP (pyoko.manage.FlushDB attribute), 14
 HELP (pyoko.manage.LoadData attribute), 14
 HELP (pyoko.manage.SchemaUpdate attribute), 14
 HELP (pyoko.manage.Shell attribute), 15
 HELP (pyoko.manage.TestGetKeys attribute), 15

I

Id (class in pyoko.fields), 10
 Integer (class in pyoko.fields), 10
 is_in_db() (pyoko.model.Model method), 17

J

JSON (pyoko.manage.DumpData attribute), 13
 JSON (pyoko.manage.LoadData attribute), 14

K

key (pyoko.node.LazyModel attribute), 19

L

LazyModel (class in pyoko.node), 19
 LinkProxy (class in pyoko.model), 15
 ListNode (class in pyoko.listnode), 10
 LoadData (class in pyoko.manage), 14

M

ManagementCommands (class in pyoko.manage), 14
 Model (class in pyoko.model), 15
 Model (pyoko.db.queryset.ReturnType attribute), 6
 ModelMeta (class in pyoko.modelmeta), 18
 MultipleObjectsReturned, 8
 MyEncoder (class in pyoko.lib.utils), 7

N

Node (class in pyoko.node), 19
 NoSuchObjectError, 8
 NotCompatible, 9

O

Object (pyoko.db.queryset.ReturnType attribute), 6
 ObjectDoesNotExist, 9
 objects (pyoko.model.Model attribute), 17
 or_filter() (pyoko.db.queryset.QuerySet method), 5

P

PARAMS (pyoko.manage.Command attribute), 13

parse_args() (pyoko.manage.ManagementCommands method), 14
 post_save() (pyoko.model.Model method), 17
 pprint() (in module pyoko.lib.utils), 8
 pre_save() (pyoko.model.Model method), 17
 prepare_buckets() (pyoko.manage.LoadData method), 14
 PRETTY (pyoko.manage.DumpData attribute), 13
 PRETTY (pyoko.manage.LoadData attribute), 14
 prnt() (pyoko.model.Model method), 17
 process_attributes_of_node() (pyoko.modelmeta.ModelMeta static method), 18
 process_models() (pyoko.modelmeta.ModelMeta static method), 18
 process_objects() (pyoko.modelmeta.ModelMeta static method), 18
 pyoko (module), 20
 pyoko.conf (module), 8
 pyoko.db (module), 7
 pyoko.db.connection (module), 6
 pyoko.db.queryset (module), 3
 pyoko.db.schema_update (module), 6
 pyoko.exceptions (module), 8
 pyoko.fields (module), 9
 pyoko.lib (module), 8
 pyoko.lib.py2map (module), 7
 pyoko.lib.utils (module), 7
 pyoko.listnode (module), 10
 pyoko.manage (module), 13
 pyoko.model (module), 15
 pyoko.modelmeta (module), 18
 pyoko.node (module), 19
 pyoko.registry (module), 20
 PyokoError, 9

Q

QuerySet (class in pyoko.db.queryset), 3

R

random_word() (in module pyoko.lib.utils), 8
 raw() (pyoko.db.queryset.QuerySet method), 6
 read_file() (pyoko.manage.LoadData method), 14
 read_json_per_line() (pyoko.manage.LoadData method), 14
 read_per_line() (pyoko.manage.LoadData method), 14
 read_whole_file() (pyoko.manage.LoadData method), 14
 register_model() (pyoko.registry.Registry method), 20
 Registry (class in pyoko.registry), 20
 remove() (pyoko.listnode.ListNode method), 12
 ReturnType (class in pyoko.db.queryset), 6
 row_level_access() (pyoko.model.Model static method), 17
 run() (pyoko.db.schema_update.SchemaUpdater method), 7

run() (pyoko.manage.Command method), 13
run() (pyoko.manage.DumpData method), 13
run() (pyoko.manage.FindDuplicateKeys method), 14
run() (pyoko.manage.FlushDB method), 14
run() (pyoko.manage.LoadData method), 14
run() (pyoko.manage.SchemaUpdate method), 14
run() (pyoko.manage.Shell method), 15
run() (pyoko.manage.TestGetKeys method), 15

S

save() (pyoko.model.Model method), 18
save_obj() (pyoko.manage.LoadData method), 14
SchemaUpdate (class in pyoko.manage), 14
SchemaUpdater (class in pyoko.db.schema_update), 6
search_on() (pyoko.db.queryset.QuerySet method), 6
set_data() (pyoko.model.Model method), 18
set_params() (pyoko.db.queryset.QuerySet method), 6
Settings (class in pyoko.conf), 8
Shell (class in pyoko.manage), 14
SimpleChoicesManager (class in pyoko.lib.utils), 7
SimpleRiakFileManager (class in pyoko.lib.utils), 8
SmartFormatter (class in pyoko.manage), 15
Solr (pyoko.db.queryset.ReturnType attribute), 6
solr() (pyoko.db.queryset.QuerySet method), 6
solr_type (pyoko.fields.Boolean attribute), 9
solr_type (pyoko.fields.Date attribute), 9
solr_type (pyoko.fields.DateTime attribute), 9
solr_type (pyoko.fields.File attribute), 9
solr_type (pyoko.fields.Float attribute), 10
solr_type (pyoko.fields.Id attribute), 10
solr_type (pyoko.fields.Integer attribute), 10
solr_type (pyoko.fields.String attribute), 10
solr_type (pyoko.fields.Text attribute), 10
solr_type (pyoko.fields.TimeStamp attribute), 10
store_file() (pyoko.lib.utils.SimpleRiakFileManager method), 8
String (class in pyoko.fields), 10

T

TestGetKeys (class in pyoko.manage), 15
Text (class in pyoko.fields), 10
TimeStamp (class in pyoko.fields), 10
to_camel() (in module pyoko.lib.utils), 8
TREE (pyoko.manage.DumpData attribute), 13
TREE (pyoko.manage.LoadData attribute), 14

U

un_camel() (in module pyoko.lib.utils), 8
un_camel_id() (in module pyoko.lib.utils), 8

V

validate() (pyoko.fields.BaseField method), 9
ValidationError, 9

verbose_name (pyoko.node.LazyModel attribute), 19

W

wait_for_schema_creation() (in module
pyoko.db.schema_update), 7
wait_for_schema_deletion() (in module
pyoko.db.schema_update), 7