
AnyBlok Documentation

Release 0.9.6

Jean-Sebastien SUZANNE

Aug 03, 2017

Contents

1	Front Matter	3
1.1	Project Homepage	3
1.2	Project Status	3
1.3	Installation	3
1.4	Unit Test	4
1.5	Dependencies	4
1.6	Contributing (hackers needed!)	4
1.7	Author	4
1.8	Contributors	4
1.9	Bugs	5
2	How to create your own application	7
2.1	Declare bloks in the entry points	8
2.2	Create Bloks	8
2.3	Create Models	12
2.4	Updating an existing Model	14
2.5	Add entries in the argparse configuration	14
2.6	Create an application	14
2.7	Generic application of AnyBlok	19
2.8	AnyBlok plugin for nosetests	20
2.9	Create the configuration file	20
2.10	Default configuration file	21
3	How to add a new Type /core	23
3.1	Difference between Core and Type	23
3.2	Declare a new Type	23
3.3	Declare a Mixin entry type	26
3.4	Declare a new Core	26
4	Environment	27
4.1	Use the current environment	27
4.2	Define a new environment type	28
5	MEMENTO	31
5.1	Blok	31
5.2	Declaration	33
5.3	Model	33

5.4	Column	37
5.5	RelationShip	40
5.6	Field	41
5.7	Mixin	43
5.8	SQL View	43
5.9	Core	44
5.10	Sharing a table between more than one model	45
5.11	Sharing a view between more than one model	46
5.12	Specific behaviour	47
5.13	Plugin	51
6	AnyBlok framework	53
6.1	anyblok module	53
6.2	anyblok.declarations module	53
6.3	anyblok.model module	58
6.4	anyblok.mapper module	59
6.5	anyblok.config module	61
6.6	anyblok.logging module	64
6.7	anyblok.imp module	66
6.8	anyblok.environment module	66
6.9	anyblok.blok module	68
6.10	anyblok.registry module	70
6.11	anyblok.migration module	77
6.12	anyblok.field module	82
6.13	anyblok.column module	85
6.14	anyblok.relationship module	114
6.15	anyblok._graphviz module	126
6.16	anyblok.scripts module	128
7	Helpers for unittest	131
7.1	TestCase	131
7.2	DBTestCase	132
7.3	BlokTestCase	133
7.4	LogCapture	134
8	Bloks	139
8.1	Blok anyblok-core	139
8.2	Blok IO	164
8.3	Blok IO CSV	175
8.4	Blok IO XML	183
8.5	Blok Model Auth	187
9	CHANGELOG	189
9.1	0.9.6 (2017-07-03)	189
9.2	0.9.5 (2016-12-05)	189
9.3	0.9.4 (2016-10-27)	189
9.4	0.9.3 (2016-10-12)	189
9.5	0.9.2 (2016-10-12)	190
9.6	0.9.1 (2016-10-3)	190
9.7	0.9.0 (2016-07-11)	190
9.8	0.8.5 (2016-06-20)	191
9.9	0.8.4 (2016-06-14)	191
9.10	0.8.3 (2016-04-18)	191
9.11	0.8.2 (2016-04-06)	191
9.12	0.8.1 (2016-03-15)	191

9.13	0.8.0 (2016-02-05)	191
9.14	0.7.2 (2016-01-14)	192
9.15	0.7.1 (2016-01-08)	192
9.16	0.7.0 (2016-01-07)	192
9.17	0.6.0 (2016-01-07)	193
9.18	0.5.2 (2015-09-28)	193
9.19	0.5.1 (2015-08-29)	193
9.20	0.5.0 (2015-08-28)	193
9.21	0.4.1 (2015-07-22)	195
9.22	0.4.0 (2015-06-21)	195
9.23	0.3.5 (2015-05-10)	196
9.24	0.3.4 (2015-05-10)	196
9.25	0.3.3 (2015-05-04)	196
9.26	0.3.2 (2015-05-04)	196
9.27	0.3.1 (2015-05-04)	196
9.28	0.3.0 (2015-05-03)	196
9.29	0.2.12 (2015-04-29)	197
9.30	0.2.11 (2015-04-26)	197
9.31	0.2.10 (2015-04-26)	197
9.32	0.2.9 (2015-04-18)	197
9.33	0.2.8 (2015-04-16)	197
9.34	0.2.7 (2015-04-15)	197
9.35	0.2.6 (2015-04-11)	198
9.36	0.2.5 (2015-03-23)	198
9.37	0.2.3 (2015-03-23)	198
9.38	0.2.2 (2015-03-15)	198
9.39	0.2.0 (2015-02-13)	198
9.40	0.1.3 (2015-02-03)	199
9.41	0.1.2 (2015-02-02)	199
9.42	0.1.1 (2015-01-23)	199
9.43	0.1.0 (2015-01-23)	199
10	ROADMAP	201
10.1	To implement	201
10.2	Library to include	201
10.3	Functionnality which need a sprint	201
11	Mozilla Public License Version 2.0	205
11.1	1. Definitions	205
11.2	2. License Grants and Conditions	207
11.3	3. Responsibilities	208
11.4	4. Inability to Comply Due to Statute or Regulation	209
11.5	5. Termination	209
11.6	6. Disclaimer of Warranty	209
11.7	7. Limitation of Liability	210
11.8	8. Litigation	210
11.9	9. Miscellaneous	210
11.10	10. Versions of the License	210
11.11	Exhibit A - Source Code Form License Notice	211
11.12	Exhibit B - "Incompatible With Secondary Licenses" Notice	211
12	Indices and tables	213
	Python Module Index	215

AnyBlok is a Python framework allowing to create highly dynamic and modular applications on top of SQLAlchemy. Applications are made of “bloks” that can be installed, extended, replaced, upgraded or uninstalled. Bloks can provide SQL Models, Column types, Fields, Mixins, SQL views, or plain Python code unrelated to the database. Models can be dynamically customized, modified, or extended without strong dependencies between them, just by adding new bloks. Bloks are declared using *setuptools* entry-points.

AnyBlok is released under the terms of the *Mozilla Public License*.

Contents

- *Front Matter*
 - *Project Homepage*
 - *Project Status*
 - *Installation*
 - *Unit Test*
 - *Dependencies*
 - *Contributing (hackers needed!)*
 - *Author*
 - *Contributors*
 - *Bugs*

Information about the AnyBlok project.

Project Homepage

AnyBlok is hosted on [github](#) - the main project page is at <http://github.com/AnyBlok/AnyBlok> or <http://code.anyblok.org>. Source code is tracked here using [GIT](#).

Releases and project status are available on Pypi at <http://pypi.python.org/pypi/anyblok>.

The most recent published version of this documentation should be at <http://doc.anyblok.org>.

Project Status

AnyBlok is currently in alpha status and is expected to be fairly stable. Users should take care to report bugs and missing features on an as-needed basis. It should be expected that the development version may be required for proper implementation of recently repaired issues in between releases; the latest master is always available at <https://github.com/AnyBlok/AnyBlok/archive/master.zip>.

Installation

Install released versions of AnyBlok from the Python package index with [pip](#) or a similar tool:

```
pip install anyblok
```

Installation via source distribution is via the `setup.py` script:

```
python setup.py install
```

Installation will add the `anyblok` commands to the environment.

Note: AnyBlok use Python version `>= 3.3`

Unit Test

Run the framework test with `nose`:

```
pip install nose
nosetests anyblok/tests
```

Run all the installed bloks:

```
anyblok_nose -c config.file.cfg
```

Run the blok tests at the installation:

```
anyblok_updatedb -c config.file.cfg --install_bloks myblok --test-blok-at-install
```

AnyBlok is tested using [Travis](#)

Dependencies

AnyBlok works with **Python 3.3** and later. The install process will ensure that [SQLAlchemy](#), [Alembic](#), [SQLAlchemy-Utils](#) are installed, in addition to other dependencies.

AnyBlok works with SQLAlchemy from version **1.0.11**, Alembic from version **0.8.4** and SQLAlchemy-Utils from version **0.31.4**. The latest version of them is strongly recommended.

Contributing (hackers needed!)

Anyblok is at a very early stage, feel free to fork, talk with core dev, and spread the word!

Author

Jean-Sébastien Suzanne

Contributors

[Anybox](#) team:

- Georges Racinet
- Christophe Combelles
- Jean-Sébastien Suzanne
- Florent Jouatte

- Simon André
- Pierre Verkest

other:

- Sébastien Chazallet
- Franck Bret

Bugs

Bugs and feature enhancements to AnyBlok should be reported on the [Issue tracker](#).

Contents

- *How to create your own application*
 - *Declare bloks in the entry points*
 - *Create Bloks*
 - *Create Models*
 - *Updating an existing Model*
 - *Add entries in the argparse configuration*
 - *Create an application*
 - *Generic application of AnyBlok*
 - *AnyBlok plugin for nosetests*
 - *Create the configuration file*
 - *Default configuration file*

How to create your own application

This first part introduces how to create an application with his code. Why do we have to create an application ? Because AnyBlok is just a framework not an application.

The goal is that more than one application can use the same database for different usage. The web server needs to give access to the user, but a profiler needs another access with another access rule, or another application needs to provide one part of the fonctionnalities.

We will write a simple application that connects to a new empty database:

- **Employee**
 - name: employee's name
 - office (Room): the room where the employee works
 - position: employee position (manager, developer...)
- **Room**
 - number: describe the room in the building
 - address: postal address
 - employees: men and women working in that room
- **Address**
 - street
 - zipcode
 - city
 - rooms: room list
- **Position**
 - name: position name

Declare bloks in the entry points

A blok must be declared in a `setuptools` entry point named **bloks**.

File tree:

```
WorkBlok
- setup.py
```

We declare 4 bloks in the `setup.py` file that we will define explain after:

```
bloks = [
    'office=exampleblok.office_blok:OfficeBlok',
    'employee=exampleblok.employee_blok:EmployeeBlok',
    'position=exampleblok.position_blok:PositionBlok',
    'employee-position=exampleblok.employee_position_blok:EmployeePositionBlok',
],

setup(
    # (...)
    entry_points={
        'bloks': bloks,
    },
)
```

Create Bloks

A blok contains Declarations such as:

- Model: a Python class usable by the application and linked in the registry
- Mixin: a Python class to extend Model
- Column: a Python class, describing an sql column type
- Relationship: a Python class, allowing to surh on the join on the model data
- ...

The blok name must be declared in the blok group of the `setup.py` file of the distribution as explain before.

And the blok must inherit the Blok class of anyblok in the `__init__.py` file of a package:

```
from anyblok.blok import Blok

class MyFirstBlok(Blok):
    """ This is valid blok """
```

The blok class must be in the init file of the package so that all modules and sub-packages which have declarations have to be imported by anyblok, in the `import_declaration_module` method

Office blok

File tree:

```
office_blok
- __init__.py
- office.py
```

__init__.py file:

```
from anyblok.blok import Blok

class OfficeBlok(Blok):

    version = '1.0.0'

    def install(self):
        """ method called at blok installation time """
        address = self.registry.Address.insert(street='14-16 rue Soleillet',
                                                zip='75020', city='Paris')
        self.registry.Room.insert(number=308, address=address)

    def update(self, latest_version):
        if latest_version is None:
            self.install()

    @classmethod
    def import_declaration_module(cls):
        from . import office # noqa

# office.py describe the models Address and Room
```

Position blok

File tree:

```
position_blok
- __init__.py
- position.py
```

__init__.py file:

```
from anyblok.blok import Blok

class PositionBlok(Blok):

    version = '1.0.0'

    def install(self):
        self.registry.Position.multi_insert({'name': 'CTO'},
                                            {'name': 'CEO'},
                                            {'name': 'Administrative Manager'},
                                            {'name': 'Project Manager'},
                                            {'name': 'Developer'})

    def update(self, latest_version):
        if latest_version is None:
            self.install()

    @classmethod
    def import_declaration_module(cls):
        from . import position # noqa

# position.py describe the model Position
```

Employee blok

Some bloks can have requirements. Each blok define its dependencies:

- required: required bloks must be loaded before
- optional: If the blok exists, optional bloks will be loaded

A blok can be declared as `autoinstall` if the blok is not installed upon the loading of the registry, then this blok will be loaded and installed.

File tree:

```
employee_blok
- __init__.py
- config.py
- employee.py
```

`__init__.py` file:

```
from anyblok.blok import Blok

class EmployeeBlok(Blok):

    version = '1.0.0'
    autoinstall = True

    required = [
        'office',
    ]

    optional = [
        'position',
    ]

    def install(self):
        room = self.registry.Room.query().filter(
            self.registry.Room.number == 308).first()
        employees = [dict(name=employee, room=room)
                      for employee in ('Georges Racinet',
                                      'Christophe Combelles',
                                      'Sandrine Chaufournaïs',
                                      'Pierre Verkest',
                                      'Franck Bret',
                                      "Simon André",
                                      'Florent Jouatte',
                                      'Clovis Nzouendjou',
                                      u"Jean-Sébastien Suzanne")]
        self.registry.Employee.multi_insert(*employees)

    def update(self, latest_version):
        if latest_version is None:
            self.install()

    @classmethod
    def import_declaration_module(cls):
        from . import config # noqa
        from . import employee # noqa
```



```
# employee.py describe the model Employee
```

EmployeePosition blok:

Some bloks can be installed when other bloks are installed, they are called conditional bloks.

File tree:

```
employee_position_blok
- __init__.py
- employee.py
```

__init__.py file:

```
from anyblok.blok import Blok

class EmployeePositionBlok(Blok):

    version = '1.0.0'
    priority = 200

    conditional = [
        'employee',
        'position',
    ]

    def install(self):
        Employee = self.registry.Employee

        position_by_employee = {
            'Georges Racinet': 'CTO',
            'Christophe Combelles': 'CEO',
            'Sandrine Chaufournais': u"Administrative Manager",
            'Pierre Verkest': 'Project Manager',
            'Franck Bret': 'Project Manager',
            u"Simon André": 'Developer',
            'Florent Jouatte': 'Developer',
            'Clovis Nzouendjou': 'Developer',
            u"Jean-Sébastien Suzanne": 'Developer',
        }

        for employee, position in position_by_employee.items():
            Employee.query().filter(Employee.name == employee).update({
                'position_name': position})

    def update(self, latest_version):
        if latest_version is None:
            self.install()

    @classmethod
    def import_declaration_module(cls):
        from . import employee # noqa
```

Warning: There are no strong dependencies between conditional blok and bloks, so the priority number of the conditional blok must be bigger than bloks defined in the *conditional* list. Bloks are loaded by dependencies and

priorities so a blok with small dependency/priority will be loaded before a blok with an higher dependency/priority.

Create Models

The Model must be added under the Model node of the declaration with the class decorator `Declarations.register`:

```
from anyblok import Declarations

@Declarations.register(Declarations.Model)
class AAnyBlokModel:
    """ The first Model of our application """
```

There are two types of Model:

- SQL: Create a table in the database (inherit `SqlBase` and `Base`)
- Non SQL: No table but the model exists in the registry and can be used (inherits `Base`).

`SqlBase` and `Base` are core models. Directly calling them is not allowed. But they are inheritable and each subclass is propagated to all the anyblok models. This example uses `insert` and `multi_insert` added by the `anyblok-core` blok.

An SQL model can define columns:

```
from anyblok import Declarations
from anyblok.column import String

register = Declarations.register
Model = Declarations.Model

@register(Model)
class ASQLModel:

    acolumn = String(label="The first column", primary_key=True)
```

Warning: Any SQL Model must have a primary key composed with one or more columns.

Warning: The table name depends on the registry tree. Here the table is `asqlmodel`. If a new model is defined under `ASQLModel` (example `UnderModel: asqlcolumn_undermodel`), the registry model will be stored as `Model.ASQLModel.UnderModel`

office_blok.office:

```
from anyblok import Declarations
from anyblok.column import Integer, String
from anyblok.relationship import Many2One

register = Declarations.register
Model = Declarations.Model
```

```

@register(Model)
class Address:

    id = Integer(label="Identifier", primary_key=True)
    street = String(label="Street", nullable=False)
    zip = String(label="Zip", nullable=False)
    city = String(label="City", nullable=False)

    def __str__(self):
        return "%s %s %s" % (self.street, self.zip, self.city)

@register(Model)
class Room:

    id = Integer(label="Identifier", primary_key=True)
    number = Integer(label="Number of the room", nullable=False)
    address = Many2One(label="Address", model=Model.Address, nullable=False,
                        one2many="rooms")

    def __str__(self):
        return "Room %d at %s" % (self.number, self.address)

```

The relationships can also define the opposite relation. Here the address Many2One relation also declares the room One2Many relation on the Address Model

A Many2One or One2One relationship must have an existing column. The `column_name` attribute allows to choose the linked column, if this attribute is missing then the value is `"model.table".remote_column`. If the linked column does not exist, the relationship creates the column with the same type as the `remote_column`.

position_blok.position:

```

from anyblok import Declarations
from anyblok.column import String

register = Declarations.register
Model = Declarations.Model

@register(Model)
class Position:

    name = String(label="Position", primary_key=True)

    def __str__(self):
        return self.name

```

employee_blok.employee:

```

from anyblok import Declarations
from anyblok.column import String
from anyblok.relationship import Many2One

register = Declarations.register
Model = Declarations.Model

```

```
@register(Model)
class Employee:

    name = String(label="Number of the room", primary_key=True)
    room = Many2One(label="Office", model=Model.Room, one2many="employees")

    def __str__(self):
        return "%s in %s" % (self.name, self.room)
```

Updating an existing Model

If you create 2 models with the same declaration position and the same name, the second model will subclass the first model. The two models will be merged to get the real model

employee_position_blok.employee:

```
from anyblok import Declarations
from anyblok.relationship import Many2One

register = Declarations.register
Model = Declarations.Model

@register(Model)
class Employee:

    position = Many2One(label="Position", model=Model.Position, nullable=False)

    def __str__(self):
        res = super(Employee, self).__str__()
        return "%s (%s)" % (res, self.position)
```

Add entries in the argparse configuration

Some applications may require options. Options are grouped by category. And the application chooses the option category to display.

employee_blok.config:

```
from anyblok.config import Configuration

@Configuration.add('message', label="This is the group message")
def add_interpreter(parser, configuration):
    parser.add_argument('--message-before', dest='message_before')
    parser.add_argument('--message-after', dest='message_after')
```

Create an application

The application can be a simple script or a setuptools script. For a setuptools script, add this in the `setup.py`:

```

setup(
    ...
    entry_points={
        'console_scripts': ['exampleblok=exampleblok.scripts:exampleblok'],
        'bloks': bloks,
    },
)

```

The script must display:

- the provided message_before
- the lists of the employee by address and by room
- the provided message_after

scripts.py:

```

import anyblok
from logging import getLogger
from anyblok.config import Configuration

logger = getLogger()

def exampleblok():
    # Initialise the application, with a name and a version number
    # select the groupe of options to display
    # return a registry if the database are selected
    registry = anyblok.start(
        'Example Blok', argparse_groups=['message', 'logging'])

    if not registry:
        return

    message_before = Configuration.get('message_before')
    message_after = Configuration.get('message_after')

    if message_before:
        logger.info(message_before)

    for address in registry.Address.query().all():
        for room in address.rooms:
            for employee in room.employees:
                logger.info(employee)

    if message_after:
        logger.info(message_after)

```

Display the help of your application:

```

jssuzanne:anyblok jssuzanne$ ./bin/exampleblok -h
usage: exampleblok [-h]
                    [--logging-level {NOTSET,DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                    [--logging-level-qualnames LOGGING_LEVEL_QUALNAMES [LOGGING_LEVEL_
↪QUALNAMES ...]]
                    [--logging-config-file LOGGING_CONFIGFILE]
                    [--logging-json-config-file JSON_LOGGING_CONFIGFILE]
                    [--logging-yaml-config-file YAML_LOGGING_CONFIGFILE]

```

```

        [-c CONFIGFILE] [--without-auto-migration]
        [--db-name DB_NAME] [--db-driver-name DB_DRIVER_NAME]
        [--db-user-name DB_USER_NAME] [--db-password DB_PASSWORD]
        [--db-host DB_HOST] [--db-port DB_PORT] [--db-echo]

[options] -- other arguments

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIGFILE          Relative path of the config file
  --without-auto-migration

Logging:
  --logging-level {NOTSET,DEBUG,INFO,WARNING,ERROR,CRITICAL}
  --logging-level-qualnames LOGGING_LEVEL_QUALNAMES [LOGGING_LEVEL_QUALNAMES ...]
                        Limit the log level on a qualnames list
  --logging-config-file LOGGING_CONFIGFILE
                        Relative path of the logging config file
  --logging-json-config-file JSON_LOGGING_CONFIGFILE
                        Relative path of the logging config file (json). Only
                        if the logging config file doesn't filled
  --logging-yaml-config-file YAML_LOGGING_CONFIGFILE
                        Relative path of the logging config file (yaml). Only
                        if the logging and json config file doesn't filled

Database:
  --db-name DB_NAME      Name of the database
  --db-driver-name DB_DRIVER_NAME
                        the name of the database backend. This name will
                        correspond to a module in sqlalchemy/databases or a
                        third party plug-in
  --db-user-name DB_USER_NAME
                        The user name
  --db-password DB_PASSWORD
                        database password
  --db-host DB_HOST      The name of the host
  --db-port DB_PORT      The port number
  --db-echo

```

Create an empty database and call the script:

```

jssuzanne:anyblok jssuzanne$ createdb anyblok
jssuzanne:anyblok jssuzanne$ ./bin/exampleblok -c anyblok.cfg --message-before "Get_
↪the employee ..." --message-after "End ..."
2014-1129 10:54:27 INFO - anyblok:root - Registry.load
2014-1129 10:54:27 INFO - anyblok:anyblok.registry - Blok 'anyblok-core' loaded
2014-1129 10:54:27 INFO - anyblok:anyblok.registry - Assemble 'Model' entry
2014-1129 10:54:27 INFO - anyblok:alembic.migration - Context impl PostgresqlImpl.
2014-1129 10:54:27 INFO - anyblok:alembic.migration - Will assume transactional DDL.
2014-1129 10:54:27 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↪'system_cache_id_seq' as owned by integer column 'system_cache(id)', assuming_
↪SERIAL and omitting
2014-1129 10:54:27 INFO - anyblok:anyblok.registry - Initialize 'Model' entry
2014-1129 10:54:27 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok_
↪- Install the blok 'anyblok-core'
2014-1129 10:54:27 INFO - anyblok:root - Registry.reload
2014-1129 10:54:27 INFO - anyblok:root - Registry.load
2014-1129 10:54:27 INFO - anyblok:anyblok.registry - Blok 'anyblok-core' loaded

```

```

2014-1129 10:54:27 INFO - anyblok:anyblok.registry - Blok 'office' loaded
2014-1129 10:54:27 INFO - anyblok:anyblok.registry - Assemble 'Model' entry
2014-1129 10:54:27 INFO - anyblok:alembic.migration - Context impl PostgresqlImpl.
2014-1129 10:54:27 INFO - anyblok:alembic.migration - Will assume transactional DDL.
2014-1129 10:54:27 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'address_id_seq' as owned by integer column 'address(id)', assuming SERIAL and
↳ omitting
2014-1129 10:54:27 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'system_cache_id_seq' as owned by integer column 'system_cache(id)', assuming
↳ SERIAL and omitting
2014-1129 10:54:27 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'room_id_seq' as owned by integer column 'room(id)', assuming SERIAL and omitting
2014-1129 10:54:27 INFO - anyblok:anyblok.registry - Initialize 'Model' entry
2014-1129 10:54:28 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Install the blok 'office'
2014-1129 10:54:28 INFO - anyblok:root - Registry.reload
2014-1129 10:54:28 INFO - anyblok:root - Registry.load
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Blok 'anyblok-core' loaded
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Blok 'office' loaded
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Blok 'position' loaded
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Assemble 'Model' entry
2014-1129 10:54:28 INFO - anyblok:alembic.migration - Context impl PostgresqlImpl.
2014-1129 10:54:28 INFO - anyblok:alembic.migration - Will assume transactional DDL.
2014-1129 10:54:28 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'address_id_seq' as owned by integer column 'address(id)', assuming SERIAL and
↳ omitting
2014-1129 10:54:28 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'system_cache_id_seq' as owned by integer column 'system_cache(id)', assuming
↳ SERIAL and omitting
2014-1129 10:54:28 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'room_id_seq' as owned by integer column 'room(id)', assuming SERIAL and omitting
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Initialize 'Model' entry
2014-1129 10:54:28 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Install the blok 'position'
2014-1129 10:54:28 INFO - anyblok:root - Registry.reload
2014-1129 10:54:28 INFO - anyblok:root - Registry.load
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Blok 'anyblok-core' loaded
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Blok 'office' loaded
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Blok 'position' loaded
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Blok 'employee' loaded
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Assemble 'Model' entry
2014-1129 10:54:28 INFO - anyblok:alembic.migration - Context impl PostgresqlImpl.
2014-1129 10:54:28 INFO - anyblok:alembic.migration - Will assume transactional DDL.
2014-1129 10:54:28 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'system_cache_id_seq' as owned by integer column 'system_cache(id)', assuming
↳ SERIAL and omitting
2014-1129 10:54:28 INFO - anyblok:anyblok.registry - Initialize 'Model' entry
2014-1129 10:54:29 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Install the blok 'employee'
2014-1129 10:54:29 INFO - anyblok:root - Registry.reload
2014-1129 10:54:29 INFO - anyblok:root - Registry.load
2014-1129 10:54:29 INFO - anyblok:anyblok.registry - Blok 'anyblok-core' loaded
2014-1129 10:54:29 INFO - anyblok:anyblok.registry - Blok 'office' loaded
2014-1129 10:54:29 INFO - anyblok:anyblok.registry - Blok 'position' loaded
2014-1129 10:54:29 INFO - anyblok:anyblok.registry - Blok 'employee' loaded
2014-1129 10:54:29 INFO - anyblok:anyblok.registry - Blok 'employee-position' loaded
2014-1129 10:54:29 INFO - anyblok:anyblok.registry - Assemble 'Model' entry
2014-1129 10:54:29 INFO - anyblok:alembic.migration - Context impl PostgresqlImpl.

```

```

2014-1129 10:54:29 INFO - anyblok:alembic.migration - Will assume transactional DDL.
2014-1129 10:54:29 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'system_cache_id_seq' as owned by integer column 'system_cache(id)', assuming
↳ SERIAL and omitting
2014-1129 10:54:29 INFO - anyblok:alembic.autogenerate.compare - Detected added
↳ column 'employee.position_name'
2014-1129 10:54:29 WARNING - anyblok:anyblok.migration - (IntegrityError) column
↳ "position_name" contains null values
'ALTER TABLE employee ALTER COLUMN position_name SET NOT NULL' {}
2014-1129 10:54:29 INFO - anyblok:anyblok.registry - Initialize 'Model' entry
2014-1129 10:54:29 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Install the blok 'employee-position'
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'anyblok-core'
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'office'
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'position'
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'employee'
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'employee-position'
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Get the employee ...
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Sandrine Chaufournaïs in Room
↳ 308 at 14-16 rue Soleillet 75020 Paris (Administrative Manager)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Christophe Combelles in Room
↳ 308 at 14-16 rue Soleillet 75020 Paris (CEO)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Clovis Nzouendjou in Room 308
↳ at 14-16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Florent Jouatte in Room 308
↳ at 14-16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Simon André in Room 308 at 14-
↳ 16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Jean-Sébastien Suzanne in
↳ Room 308 at 14-16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Georges Racinet in Room 308
↳ at 14-16 rue Soleillet 75020 Paris (CTO)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Pierre Verkest in Room 308 at
↳ 14-16 rue Soleillet 75020 Paris (Project Manager)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Franck Bret in Room 308 at 14-
↳ 16 rue Soleillet 75020 Paris (Project Manager)
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - End ...

```

The registry is loaded twice:

- The first load installs the bloks anyblok-core, office, position and employee
- The second load installs the conditional blok employee-position and runs a migration to add the field employee_name

Call the script again:

```

jssuzanne:anyblok jssuzanne$ ./bin/exampleblok -c anyblok.cfg --message-before "Get
↳ the employee ..." --message-after "End ..."
2014-1129 10:57:52 INFO - anyblok:root - Registry.load
2014-1129 10:57:52 INFO - anyblok:anyblok.registry - Blok 'anyblok-core' loaded
2014-1129 10:57:52 INFO - anyblok:anyblok.registry - Blok 'office' loaded
2014-1129 10:57:52 INFO - anyblok:anyblok.registry - Blok 'position' loaded
2014-1129 10:57:52 INFO - anyblok:anyblok.registry - Blok 'employee' loaded

```



```

2014-1129 10:57:52 INFO - anyblok:anyblok.registry - Blok 'employee-position' loaded
2014-1129 10:57:52 INFO - anyblok:anyblok.registry - Assemble 'Model' entry
2014-1129 10:57:52 INFO - anyblok:alembic.migration - Context impl PostgresqlImpl.
2014-1129 10:57:52 INFO - anyblok:alembic.migration - Will assume transactional DDL.
2014-1129 10:57:52 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named
↳ 'system_cache_id_seq' as owned by integer column 'system_cache(id)', assuming
↳ SERIAL and omitting
2014-1129 10:57:52 INFO - anyblok:alembic.autogenerate.compare - Detected NOT NULL on
↳ column 'employee.position_name'
2014-1129 10:57:52 INFO - anyblok:anyblok.registry - Initialize 'Model' entry
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'anyblok-core'
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'office'
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'position'
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'employee'
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok
↳ - Load the blok 'employee-position'
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Get the employee ...
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Sandrine Chaufournaïs in Room
↳ 308 at 14-16 rue Soleillet 75020 Paris (Administrative Manager)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Christophe Combelles in Room
↳ 308 at 14-16 rue Soleillet 75020 Paris (CEO)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Clovis Nzouendjou in Room 308
↳ at 14-16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Florent Jouatte in Room 308
↳ at 14-16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Simon André in Room 308 at 14-
↳ 16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Jean-Sébastien Suzanne in
↳ Room 308 at 14-16 rue Soleillet 75020 Paris (Developer)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Georges Racinet in Room 308
↳ at 14-16 rue Soleillet 75020 Paris (CTO)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Pierre Verkest in Room 308 at
↳ 14-16 rue Soleillet 75020 Paris (Project Manager)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Franck Bret in Room 308 at 14-
↳ 16 rue Soleillet 75020 Paris (Project Manager)
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - End ...

```

The registry is loaded only once, because the bloks are already installed

Generic application of AnyBlok

Anyblok provides some console script to help :

- anyblok_createdb
- anyblok_updatedb
- anyblok_interpreter .. note:

```
if IPython is in the sys.modules then the interpreter is an IPython interpreter
```

- anyblok_nose (nose test)

TODO: I know it's not a setuptools documentation but it could be kind to show a complete minimalist example of *setup.py* with requires (to anyblok). We could also display the full tree from root

A direct link to download the full working example.

AnyBlok plugin for nosetests

You can test your blocks in your anyblok distribution with nose. use the option `--with-anyblok-bloks`. The plugin load the BlokManager et the RegistryManager after load the coverage plugin.

Create the configuration file

The configuration file allow to load all the initialisation variable:

```
[AnyBlok]
key = value
```

You can extend an existing config file:

```
[AnyBlok]
extend = ``path of the configfile``
```

The logging configuration are also loaded, see [logging configuration file format](#):

```
[AnyBlok]
logging_configfile = ``name of the config file``
# json_logging_configfile = logging config file write with json
# yaml_logging_configfile = logging config file write with yaml

loggers]
keys=root,anyblok

[handlers]
keys=consoleHandler

[formatters]
keys=consoleFormatter

[logger_root]
level=INFO
handlers=consoleHandler

[logger_anyblok]
level=INFO
handlers=consoleHandler
qualname=anyblok
propagate=1

[handler_consoleHandler]
class=StreamHandler
level=INFO
formatter=consoleFormatter
args=(sys.stdout,)
```

```
[formatter_consoleFormatter]
class=anyblok.logging.consoleFormatter
format=%(database)s: %(levelname)s - %(message)s
datefmt=
```

Default configuration file

You can define default *system* or *user* configuration file in fonction of your *OS*:

- *linux*
 - *system*: /etc/xdg/AnyBlok/conf.cfg
 - *user*: /home/user name/.config/AnyBlok/conf.cfg
- *mac os x*
 - *system*: /Library/Application Support/AnyBlok/conf.cfg
 - *user*: /Users/user name/Library/Application Support/AnyBlok/conf.cfg

Note: Works also for *windows*, See <https://pypi.python.org/pypi/appdirs>. The entry used are:

- *system*: site_config_dir
- *user*: user_config_dir

Theses configuration files are load before the specific configuration file. If the the configuration file does not exist then it will not raise error

Contents

- *How to add a new Type /core*
 - *Difference between Core and Type*
 - *Declare a new Type*
 - *Declare a Mixin entry type*
 - *Declare a new Core*

How to add a new Type /core

Type and Core are both Declarations.

Difference between Core and Type

Core is also an Entry Type. But it is a particular entry Type. Core is used to define low level at the entry Type. For example the Core.Base is the low level at all the Model. Modify the behaviours of the Core.Base is equal to modify the behaviours of all the Model.

this is the inheritance model of the Model Type

Entry Type	inheritance Types	Core
Model	Model / Mixin	Base

Declare a new Type

The declaration of new Type, is declarations of a new type of declaration. The known Type declarations are:

- Model
- Mixin
- Core
- AuthorizationPolicyAssociation

This is an example to declare new entry Type:

```
from anyblok import Declarations

@Declarations.add_declaration_type()
class MyType:
```

```
@classmethod
def register(cls, parent, name, cls_, **kwargs):
    ...

    @classmethod
    def unregister(cls, child, cls_):
        ...
```

The Type must implement:

Method name	Description
register	This classmethod describe what append when a a declaration is done by he decorator <code>Declarations.register</code>
unregister	This classmethod describe what append when an undeclaration is done.

The `add_declaration_type` can define the arguments:

Argument's name	Description
isAnEntry	Boolean Define if the new <code>Type</code> is an entry, depend of the installation or not of the bloks
assemble	<p>Only for the entry “Type“ Waiting the name of the classmethod which make the action to group and create a new class with the complete inheritance tree:</p> <pre>@add_declaration_type(isAnEntry=True, assemble='assemble →')</pre> <pre>class MyType: ... @classmethod def assemble(cls, registry): ...</pre> <div> <p>Warning:</p> <p>reg- istry is the reg- istry of the database</p> </div>
initialize	<p>Only for the entry “Type“ Waiting the name of the classmethod which make the action to initialize the registry:</p> <pre>@add_declaration_type(isAnEntry=True, initialize= →'initialize')</pre> <pre>class MyType: ... @classmethod def initialize(cls, registry): ...</pre> <div> <p>Warning:</p> <p>reg- istry is the reg- istry of the database</p> </div>

Declare a Mixin entry type

Mixin is a Type to add behaviours, it is not a particular Type. But it is always very interesting to use it.

AnyBlok had already a Mixin Type for the Model Type. The Mixin Type must not be the same for all the entry Type, then Model inherit only other Model or Declarations.Mixin. If you add an another Declarations . AnotherMixin then Model won't inherit this Mixin Type.

The new Mixin Type is easy to add:

```
from anyblok import Declarations
from anyblok.mixin import MixinType

@Declarations.add_declaration_type(isAnEntry=True)
class MyMixin(MixinType):
    pass
```

Declare a new Core

The definition of a Core and the Declaration is in different parts

Declarations of a new Core:

```
from anyblok.registry import RegistryManager

RegistryManager.declare_core('MyCore')
```

Definition or register of an overload of the Core declaration:

```
from anyblok import Declarations

@Declarations.register(Declarations.Core)
class MyCore:
    ...
```

The declaration must be done in the application, not in the blok. The is only done in the blok.

Warning: Core can't inherit Model, Mixin or other Type

Contents

- *Environmmment*
 - *Use the current environment*
 - * *Generic use*
 - * *Use in a Model*
 - *Define a new environment type*

CHAPTER 4

Environment

Environment stocks contextual variable. by default the environment is stocked in the current Thread.

Use the current environment

The environment can be used wherever in the code.

Generic use

To get or set variable in environment, you must import the `EnvironmentManager`:

```
from anyblok.environment import EnvironmentManager
```

Set a variable:

```
EnvironmentManager.set('my variable name', OneValue)
```

Get a variable:

```
EnvironmentManager.get('my variable name', default=OneDefaultValue)
```

Use in a Model

A facility are add in the `registry_base`. This class is inherited by all the model.

Get the environment in `Model` method or classmethod:

```
self.Env # or cls.Env
```

Set a variable:

```
self.Env.set('my variable name', OneValue)
```

Get a variable:

```
self.Env.get('my variable name', default=OneDefaultValue)
```

Define a new environment type

If you do not want to stock the environment in the `Thread`, you must implement a new type of environment.

This type is a simple class which have theses class methods:

- `scoped_function_for_session`
- `setter`
- `getter`

```
MyEnvironmentClass:

    @classmethod
    def scoped_function_for_session(cls):
        ...

    @classmethod
    def setter(cls, key, value):
        ...

    @classmethod
    def getter(cls, key, default):
        ...
        return value
```

Declare your class as the Environment class:

```
EnvironmentManager.define_environment_cls(MyEnvironmentClass)
```

The classmethod `scoped_function_for_session` is passed at SQLAlchemy `scoped_session` function see

Contents

- *MEMENTO*
 - *Blok*
 - *Declaration*
 - *Model*
 - * *Non SQL Model*
 - * *SQL Model*
 - * *View Model*
 - *Column*

- *Relationship*
- *Field*
- *Mixin*
- *SQL View*
- *Core*
 - * *Base*
 - * *SqlBase*
 - * *SqlViewBase*
 - * *Query*
 - * *Session*
 - * *InstrumentedList*
- *Sharing a table between more than one model*
- *Sharing a view between more than one model*
- *Specific behaviour*
 - * *Column encryption*
 - * *Cache*
 - * *Event*
 - * *Hybrid method*
 - * *Pre-commit hook*
 - * *Aliased*
 - * *Get the registry*
 - * *Get the current environment*
 - * *Initialize some data by entry point*
- *Plugin*
 - * *Define a new plugin*

Anyblok mainly depends on:

- Python 3.2+
- [SQLAlchemy](#)
- [Alembic](#)

Blok

A blok is a collection of source code files. These files are loaded in the registry only if the blok state is installed.

To declare a blok you have to:

1. Declare a Python package:

```
The name of the module is not really significant
--> Just create an ``__init__.py`` file
```

2. Declare a blok class in the `__init__.py` of the Python package:

```
from anyblok.blok import Blok

class MyBlok(Blok):
    """ Short description of the blok """
    ...
    version = '1.0.0'
```

Here are the available attributes for the blok:

Attribute	Description
<code>__doc__</code>	Short description of the blok (in the docstring)
<code>version</code>	the version of the blok (required because no value by default)
<code>autoinstall</code>	boolean, if <code>True</code> this blok is automatically installed
<code>priority</code>	installation order of the blok to installation
<code>readme</code>	Path of the 'readme' file of the blok, by default <code>README.rst</code>
<code>required</code>	List of the required dependancies for install
<code>optional</code>	List of the optional dependencies, their are installed if they are found
<code>conflicting</code>	List the blok which are not be installed to install this blok
<code>conditionnal</code>	If the blocks of this list are installed the this blok will be automatically installed

And the methods that define blok behaviours:

Method	Description
<code>import_declaration</code>	class method, call to import all python module which declare object from blok.
<code>reload_declaration</code>	class method, call to reload the import all the python module which declare object
<code>update</code>	Action to do when the blok is being install or updated. This method has one argument <code>latest_version</code> (None for install)
<code>uninstall</code>	Action to do when the blok is being uninstalled
<code>load</code>	Action to do when the server starts
<code>pre_migration</code>	Action to do when the blok is being installed or updated to make some specific migration, before auto migration. This method has one argument <code>latest_version</code> (None for install)
<code>post_migration</code>	Action to do when the blok is being installed or updated to make some specific migration, after auto migration. This method has one argument <code>latest_version</code> (None for install)

And some facility:

Method	Description
<code>import_file</code>	facility to import data

Note: The version 0.2.0 change the import and reload of the module python

3. Declare the entry point in the `setup.py`:

```
from setuptools import setup

setup(
    ...
    entry_points={
        'bloks': [
            'web=anyblok_web_server.bloks.web:Web',
        ],
    },
    ...
)
```

Note: The version 0.4.0, required all the declaration of the bloks on the entry point **bloks**

Declaration

In AnyBlok, everything is a declaration (Model, Mixin, ...) and you have to import the `Declarations` class:

```
from anyblok.declarations import Declarations
```

The `Declarations` has two main methods

Method name	Description
register	<p>Add the declaration in the registry This method can be used as:</p> <ul style="list-style-type: none"> A function: <pre>class Foo: pass register(`Declarations.type`, cls_ ↪=Foo)</pre> A decorator: <pre>@register(`Declarations.type`) class Foo: pass</pre>
unregister	<p>Remove an existing declaration from the registry. This method is only used as a function:</p> <pre>from ... import Foo unregister(`Declarations.type`, cls_ ↪=Foo)</pre>

Note: `Declarations.type` must be replaced by:

- Model
- ...

`Declarations.type` defines the behaviour of the `register` and `unregister` methods

Model

A Model is an AnyBlok class referenced in the registry. The registry is hierarchical. The model `Foo` is accessed by `registry.Foo` and the model `Foo.Bar` is accessed by `registry.Foo.Bar`.

To declare a Model you must use `register`:

```
from anyblok.declarations import Declarations

register = Declarations.register
Model = Declarations.Model

@register(Model):
class Foo:
    pass
```

The name of the model is defined by the name of the class (here `Foo`). The namespace of `Foo` is defined by the hierarchy under `Model`. In this example, `Foo` is in `Model`, you can access it at `Foo` by `Model.Foo`.

Warning: `Model.Foo` is not the `Foo Model`. It is an avatar of `Foo` only used for the declaration.

If you define the `Bar` model, under the `Foo` model, you should write:

```
@register(Model.Foo)
class Bar:
    """ Description of the model """
    pass
```

Note: The description is used by the model `System.Model` to describe the model

The declaration name of `Bar` is `Model.Foo.Bar`. The namespace of `Bar` in the registry is `Foo.Bar`. The namespace of `Foo` in the registry is `Foo`:

```
Foo = registry.Foo
Bar = registry.Foo.Bar
```

Some models have a table in the database. The name of the table is by default the namespace in lowercase with `.` replaced with `_`.

Note: The registry is accessible only in the method of the models:

```
@register(Model)
class Foo:

    def myMethod(self):
        registry = self.registry
        Foo = registry.Foo
```

The main goal of AnyBlok is not only to add models in the registry, but also to easily overload these models. The declaration stores the Python class in the registry. If one model already exists then the second declaration of this model overloads the first model:

```
@register(Model)
class Foo:
    x = 1

@register(Model)
class Foo:
    x = 2

-----

Foo = registry.Foo
assert Foo.x == 2
```

Here are the parameters of the `register` method for `Model`:

Param	Description
cls_	Define the real class if <code>register</code> is used as a function not as a decorator
name_	Overload the name of the class: <pre>@register(Model, name_='Bar') class Foo: pass</pre> Declarations.Bar
tablename	Overload the name of the table: <pre>@register(Model, tablename='my_table') class Foo: pass</pre>
is_sql_view	Boolean flag, which indicateis if the model is based on a SQL view
tablename	Define the real name of the table. By default the table name is the registry name without the declaration type, and with '.' replaced with '_'. This attribute is also used to map an existing table declared by a previous Model. Allowed values: <ul style="list-style-type: none"> str <pre>@register(Model, tablename='foo') class Bar: pass</pre> declaration <pre>@register(Model, tablename=Model. ↪Foo) class Bar: pass</pre>

Warning: Model can only inherit simple python class, Mixin or Model.

Non SQL Model

This is the default model. This model has no tables. It is used to organize the registry or for specific process.:

```
#register(Model)
class Foo:
    pass
```

SQL Model

A SQL Model is a simple Model with Column or Relationship. For each model, one table will be created.:

```
@register(Model)
class Foo:
    # SQL Model with mapped with the table ``foo``
```

```
id = Integer(primary_key=True)
# id is a column on the table ``foo``
```

Warning: Each SQL Model have to have got one or more primary key

In the case or you need to add some configuration in the SQLAlchemy class attrinute:

- `__table_args__`
- `__mapper_args__`

you can use the next class methods

method	description
<code>define_table_args</code>	<p>Add options for SQLAlchemy table build:</p> <ul style="list-style-type: none"> • Constraints on multiple columns • ... <pre>@classmethod def define_table_args(cls, table_args, properties): # table_args: tuple of the known # __table_args__\ # properties: properties of the assembled model # columns, registry name return my_tuple_value</pre>
<code>define_mapper_args</code>	<p>Add options for SQLAlchemy mappers build:</p> <ul style="list-style-type: none"> • polymorphisme • ... <pre>@classmethod def define_mapper_args(cls, mapper_args, properties): # table_args: dict of the known # __mapper_args__\ # properties: properties of the assembled model # columns, registry name return my_dict_value</pre>

Note: New in 0.4.0

View Model

A View Model as SQL Model. Need the declaration of Column and / or Relationship. In the register the param `is_sql_view` must be `True` and the View Model must define the `sqlalchemy_view_declaration` classmethod.:

```
@register(Model, is_sql_view=True)
class Foo:

    id = Integer(primary_key=True)
    name = String()
```

```
@classmethod
def sqlalchemy_view_declaration(cls):
    from sqlalchemy.sql import select
    Model = cls.registry.System.Model
    return select([Model.id.label('id'), Model.name.label('name')])
```

sqlalchemy_view_declaration must return a select query corresponding to the request of the SQL view.

Column

To declare a Column in a model, add a column on the table of the model.:

```
from anyblok.declarations import Declarations
from anyblok.column import Integer, String

@Declarations.register(Declaration.Model)
class MyModel:

    id = Integer(primary_key=True)
    name = String()
```

Note: Since the version 0.4.0 the Columns are not Declarations

List of the column type:

- DateTime: use datetime.datetime, with pytz for the timezone
- Decimal: use decimal.Decimal
- Float
- Time: use datetime.time
- BigInteger
- Boolean
- Date: use datetime.date
- Integer
- Interval: use datetime.timedelta
- LargeBinary
- SmallInteger
- String
- Text
- uString
- uText
- Selection
- Json

- Sequence
- Color: use `colour.Color`
- Password: use `sqlalchemy_utils.types.password.Password`
- UUID: use `uuid`
- URL: use `furl.furl`

All the columns have the following optional parameters:

Parameter	Description
label	Label of the column, If None the label is the name of column capitalized
default	<p>define a default value for this column.</p> <p>..warning:</p> <p>The default value depends of the column_↪type</p> <p>..note:</p> <p>Put the name of a classmethod to call it</p>
index	boolean flag to define whether the column is indexed
nullable	Defines if the column must be filled or not
primary_key	Boolean flag to define if the column is a primary key or not
unique	Boolean flag to define if the column value must be unique or not
foreign_key	<p>Define a foreign key on this column to another column of another model:</p> <pre>@register(Model) class Foo: id : Integer(primary_key=True)</pre> <pre>@register(Model) class Bar: id : Integer(primary_key=True) foo: Integer(foreign_key=Model.Foo. ↪use('id'))</pre> <p>If the Model Declarations doesn't exist yet, you can use the regisrty name:</p> <pre>foo: Integer(foreign_key='Model.Foo=>id ↪')</pre>
db_column_name	String to define the real column name in the table, different from the model attribute name
encrypt_key	<p>Crypt the column in the database. can take the values:</p> <ul style="list-style-type: none"> • a String ex: foo = String(encrypt_key='SecretKey') • a classmethod name on the model • True value, search in the Configuration default_encrypt_key the value, they are no default. if no value exist, an exception is raised <p>..warning:</p> <p>The python package cryptography must be_↪installed</p>

Other attribute for String and uString:

Param	Description
size	Column size in the table

Other attribute for Selection:

Param	Description
size	column size in the table
selections	dict or dict.items to give the available key with the associate label

Other attribute for Sequence:

Param	Description
size	column size in the table
code	code of the sequence
formater	formater of the sequence

Other attribute for Color:

Param	Description
size	column max size in the table

Other attribute for Password:

Param	Description
size	password max size in the table
crypt_context	see the option for the python lib passlib

..warning:

The Password column can be found **with** the query meth:

Other attribute for UUID:

Param	Description
binary	Stores a UUID in the database natively when it can and falls back to a BINARY(16) or a CHAR(32)

Relationship

To declare a Relationship in a model, add a Relationship on the table of the model.:

```
from anyblok.declarations import Declarations
from anyblok.column import Integer
from anyblok.relationship import Many2One

@Declarations.register(Declaration.Model)
class MyModel:

    id = Integer(primary_key=True)

@Declarations.register(Declaration.Model)
class MyModel2:

    id = Integer(primary_key=True)
    mymodel = Many2One(model=Declaration.Model.MyModel)
```

Note: Since the version 0.4.0 the Relationship are not Declarations

List of the Relationship type:

- One2One

- Many2One
- One2Many
- Many2Many

Parameters of a Relationship:

Param	Description
label	The label of the column
model	The remote model
remote_column	The column name on the remote model, if no remote columns are defined the remote column will be the primary column of the remote model

Parameters of the One2One field:

Param	Description
column_name	Name of the local column. If the column doesn't exist then this column will be created. If no column name then the name will be 'tablename' + '_' + name of the relationships
nullable	Indicates if the column name is nullable or not
backref	Remote One2One link with the column name

Parameters of the Many2One field:

Parameter	Description
column_name	Name of the local column. If the column doesn't exist then this column will be created. If no column name then the name will be 'tablename' + '_' + name of the relationships
nullable	Indicate if the column name is nullable or not
one2many	Opposite One2Many link with this Many2one

Parameters of the One2Many field:

Parameter	Description
primaryjoin	Join condition between the relationship and the remote column
many2one	Opposite Many2One link with this One2Many

Parameters of the Many2Many field:

Parameter	Description
join_table	many2many intermediate table between both models
m2m_remote_columns	Columns name in the join table which have got the foreign key to the remote model
local_columns	Name of the local column which holds the foreign key to the join table. If the column does not exist then this column will be created. If no column name then the name will be 'tablename' + '_' + name of the relationship
m2m_local_columns	Column name in the join table which holds the foreign key to the model
many2many	Opposite Many2Many link with this relationship

Note: Since 0.4.0, when the relationnal table is created by AnyBlok, the m2m_columns become foreign keys

Field

To declare a Field in a model, add a Field on the Model, this is not a SQL column.:

```
from anyblok.declarations import Declarations
from anyblok.field import Function
from anyblok.column import Integer
```

```

@Declarations.register(Declaration.Model)
class MyModel:

    id = Integer(primary_key=True)
    first_name = String()
    last_name = String()
    name = Function(fget='fget', fset='fset', fdel='fdel', fexpr='fexpr')

    def fget(self):
        return '{0} {1}'.format(self.first_name, self.last_name)

    def fset(self, value):
        self.first_name, self.last_name = value.split(' ', 1)

    def fdel(self):
        self.first_name = self.last_name = None

    @classmethod
    def fexpr(cls):
        return func.concat(cls.first_name, ' ', cls.last_name)

```

List of the Field type:

- Function

Parameters for Field.Function

Parameter	Description
fget	name of the method to call to get the value of field: <pre> def fget(self): return '{0} {1}'.format(self.first_ ↪name, self.last_ ↪name) </pre>
fset	name of the method to call to set the value of field: <pre> def fset(self): self.first_name, self.last_name = ↵ ↪value.split(' ', ↵ 1) </pre>
fdel	name of the method to call to del the value of field: <pre> def fdel(self): self.first_name = self.last_name = ↵ ↪None </pre>
fexp	name of the class method to call to filter on the field: <pre> @classmethod def fexp(self): return func.concat(cls.first_name, ' ↪', cls.last_name) </pre>

Mixin

A Mixin looks like a Model, but has no tables. A Mixin adds behaviour to a Model with Python inheritance:

```
@register(Mixin)
class MyMixin:

    def foo():
        pass

@register(Model)
class MyModel(Mixin.MyMixin):
    pass

-----

assert hasattr(registry.MyModel, 'foo')
```

If you inherit a mixin, all the models previously using the base mixin also benefit from the overload:

```
@register(Mixin)
class MyMixin:
    pass

@register(Model)
class MyModel(Mixin.MyMixin):
    pass

@register(Mixin)
class MyMixin:

    def foo():
        pass

-----

assert hasattr(registry.MyModel, 'foo')
```

SQL View

An SQL view is a model, with the argument `is_sql_view=True` in the register. and the classmethod `sqlalchemy_view_declaration`:

```
@register(Model)
class T1:
    id = Integer(primary_key=True)
    code = String()
    val = Integer()

@register(Model)
class T2:
    id = Integer(primary_key=True)
    code = String()
    val = Integer()
```

```
@register(Model, is_sql_view=True)
class TestView:
    code = String(primary_key=True)
    val1 = Integer()
    val2 = Integer()

    @classmethod
    def sqlalchemy_view_declaration(cls):
        """ This method must return the query of the view """
        T1 = cls.registry.T1
        T2 = cls.registry.T2
        query = select([T1.code.label('code'),
                        T1.val.label('val1'),
                        T2.val.label('val2')])
        return query.where(T1.code == T2.code)
```

Core

Core is a low level set of declarations for all the Models of AnyBlok. Core adds general behaviour to the application.

Warning: Core can not inherit Model, Mixin, Core, or other declaration type.

Base

Add a behaviour in all the Models, Each Model inherits Base. For instance, the fire method of the event come from Core.Base.

```
from anyblok import Declarations

@Declarations.register(Declarations.Core)
class Base:
    pass
```

SqlBase

Only the Models with Field, Column, Relationship inherits Core.SqlBase. For instance, the insert method only makes sense for the Model with a table.

```
from anyblok import Declarations

@Declarations.register(Declarations.Core)
class SqlBase:
    pass
```

SqlViewBase

Like SqlBase, only the SqlView inherits this Core class.

```
from anyblok import Declarations

@Declarations.register(Declarations.Core)
class SqlViewBase:
    pass
```

Query

Overloads the SQLAlchemy Query class.

```
from anyblok import Declarations

@Declarations.register(Declarations.Core)
class Query
    pass
```

Session

Overloads the SQLAlchemy Session class.

```
from anyblok import Declarations

@Declarations.register(Declarations.Core)
class Session
    pass
```

InstrumentedList

```
from anyblok import Declarations

@Declarations.register(Declarations.Core)
class InstrumentedList
    pass
```

InstrumentedList is the class returned by the Query for all the list result like:

- query.all()
- relationship list (Many2Many, One2Many)

Adds some features like getting a specific property or calling a method on all the elements of the list:

```
MyModel.query().all().foo(bar)
```

Sharing a table between more than one model

SQLAlchemy allows two methods to share a table between two or more mapping class:

- Inherit an SQL Model in a non-SQL Model:

```
@register(Model)
class Test:
    id = Integer(primary_key=True)
    name = String()

@register(Model)
class Test2(Model.Test):
    pass

-----

t1 = Test1.insert(name='foo')
assert Test2.query().filter(Test2.id == t1.id,
                             Test2.name == t1.name).count() == 1
```

- **Share the `__table__`.** AnyBlok cannot give the table at the declaration, because the table does not exist yet. But during the assembly, if the table exists and the model has the name of this table, AnyBlok directly links the table. To define the table you must use the named argument `tablename` in the `register`

```
@register(Model)
class Test:
    id = Integer(primary_key=True)
    name = String()

@register(Model, tablename=Model.Test)
class Test2:
    id = Integer(primary_key=True)
    name = String()

-----

t1 = Test1.insert(name='foo')
assert Test2.query().filter(Test2.id == t1.id,
                             Test2.name == t1.name).count() == 1
```

Warning: There are no checks on the existing columns.

Sharing a view between more than one model

Sharing a view between two Models is the merge between:

- Creating a View Model
- Sharing the same table between more than one model.

Warning: For the view you must redined the column in the Model corresponding to the view with inheritance or simple Share by `tablename`

Specific behaviour

AnyBlok implements some facilities to help developers

Column encryption

You can encrypt some columns to protect them. The python package cryptography must be installed:

```
pip install cryptography
```

Use the `encrypt_key` attribute on the column to define the key of cryptography:

```
@register(Model)
class MyModel:

    # define the specific encrypt_key
    encrypt_column_1 = String(encrypt_key='SecretKey')

    # Use the default encrypt_key
    encrypt_column_2 = String(encrypt_key=Configuration.get('default_encrypt_key'))
    encrypt_column_3 = String(encrypt_key=True)

    # Use the class method to get encrypt_key
    encrypt_column_1 = String(encrypt_key='get_encrypt_key')

    @classmethod
    def get_encrypt_key(cls):
        return 'SecretKey'
```

The encryption works for any Columns.

Cache

The cache allows to call a method more than once without having any difference in the result. But the cache must also depend on the registry database and the model. The cache of anyblok can be put on a Model, a Core or a Mixin method. If the cache is on a Core or a Mixin then the usecase depends on the registry name of the assembled model.

Use `cache` or `classmethod_cache` to apply a cache on a method:

```
from anyblok.declarations import cache, classmethod_cache
```

Warning: `cache` depend of the instance, if you want add a cache for any instance you must use `classmethod_cache`

Cache the method of a Model:

```
@register(Model)
class Foo:

    @classmethod_cache()
    def bar(cls):
        import random
        return random.random()
```

```
-----  
  
assert Foo.bar() == Foo.bar()
```

Cache the method coming from a Mixin:

```
@register(Mixin)  
class MFoo:  
  
    @classmethod_cache()  
    def bar(cls):  
        import random  
        return random.random()  
  
@register(Model)  
class Foo(Mixin.MFoo):  
    pass  
  
@register(Model)  
class Foo2(Mixin.MFoo):  
    pass  
  
-----  
  
assert Foo.bar() == Foo.bar()  
assert Foo2.bar() == Foo2.bar()  
assert Foo.bar() != Foo2.bar()
```

Cache the method coming from a Mixin:

```
@register(Core)  
class Base  
  
    @classmethod_cache()  
    def bar(cls):  
        import random  
        return random.random()  
  
@register(Model)  
class Foo:  
    pass  
  
@register(Model)  
class Foo2:  
    pass  
  
-----  
  
assert Foo.bar() == Foo.bar()  
assert Foo2.bar() == Foo2.bar()  
assert Foo.bar() != Foo2.bar()
```

Event

Simple implementation of a synchronous event for AnyBlok or SQLAlchemy:

```
@register(Model)
class Event:
    pass

@register(Model)
class Test:

    x = 0

    @listen(Model.Event, 'fireevent')
    def my_event(cls, a=1, b=1):
        cls.x = a * b

-----

registry.Event.fire('fireevent', a=2)
assert registry.Test.x == 2
```

Note: The decorated method is seen as a classmethod

This API gives:

- a decorator `listen` which binds the decorated method to the event.
- **fire method with the following parameters (Only for AnyBlok event):**
 - `event`: string name of the event
 - `*args`: positionnal arguments to pass att the decorated method
 - `**kwargs`: named argument to pass at the decorated method

It is possible to overload an existing event listener, just by overloading the decorated method:

```
@register(Model)
class Test:

    @classmethod
    def my_event(cls, **kwarg):
        res = super(Test, cls).my_event(**kwargs)
        return res * 2

-----

registry.Event.fire('fireevent', a=2)
assert registry.Test.x == 4
```

Warning: The overload does not take the `listen` decorator but the `classmethod` decorator, because the method name is already seen as an event listener

Some of the Attribute events of the Mapper events are implemented. See the SQLAlchemy ORM Events <http://docs.sqlalchemy.org/en/latest/orm/events.html#orm-events>

Hybrid method

Facility to create an SQLAlchemy hybrid method. See this page: <http://docs.sqlalchemy.org/en/latest/orm/extensions/hybrid.html#module-sqlalchemy.ext.hybrid>

AnyBlok allows to define a `hybrid_method` which can be overloaded, because the real sqlalchemy decorator is applied after assembling in the last overload of the decorated method:

```
from anyblok.declarations import hybrid_method

@register(Model)
class Test:

    @hybrid_method
    def my_hybrid_method(self):
        return ...
```

Pre-commit hook

It is possible to call specific classmethods just before the commit of the session:

```
@register(Model)
class Test:

    id = Integer(primary_key=True)
    val = Integer(default=0)

    @classmethod
    def method2call_just_before_the_commit(cls):
        pass

-----

registry.Test.precommit_hook('method2call_just_before_the_commit')
```

Aliased

Facility to create an SQL alias for the SQL query by the ORM:

```
select * from my_table the_table_alias.
```

This facility is given by SQLAlchemy, and anyblok adds this fonctionnality directly in the Model:

```
BlokAliased = registry.System.Blok.aliased()
```

Note: See this page: <http://docs.sqlalchemy.org/en/latest/orm/query.html#sqlalchemy.orm.aliased> to know the parameters of the `aliased` method

Warning: The first arg is already passed by AnyBlok

Get the registry

You can get a Model by the registry in any method of Models:

```
Model = self.registry.System.Model
assert Model.__registry_name__ == 'Model.System.Model'
```

Get the current environment

The current environment is saved in the main thread. You can add a value to the current Environment:

```
self.Env.set('My var', 'one value')
```

You can get a value from the current Environment:

```
myvalue = self.Env.get('My var', default="My default value")
```

Note: The environment is as a dict the value can be an instance of any type

Initialize some data by entry point

the entry point `anyblok.init` allow to define function, `init_function` in this example:

```
setup(
    ...
    entry_points={
        'anyblok.init': [
            'my_function=path:init_function',
        ],
    },
)
```

In the path the `init_function` must be defined:

```
def init_function(unittest=False):
    ...
```

..warning:

```
Use unittest parameter to defined if the function must be call
or not
```

Plugin

Plugin is used for the low level, it is not use in the bloks, because the model can be overload by the declaration.

Define a new plugin

A plugin can be a class or a function:

```
class MyPlugin:
    pass
```

Add the plugin definition in the configuration:

```
@Configuration.add('plugins')
def add_plugins(self, group)
    group.add_argument('--my-option', dest='plugin_name',
                       type=AnyBlokPlugin,
                       default='path:MyPlugin')
```

Use the plugin:

```
plugin = Configuration.get('plugin_name')
```

Contents

- *AnyBlok framework*
 - *anyblok module*
 - *anyblok.declarations module*
 - *anyblok.model module*
 - *anyblok.mapper module*
 - *anyblok.config module*
 - *anyblok.logging module*
 - *anyblok.imp module*
 - *anyblok.environment module*
 - *anyblok.blok module*
 - *anyblok.registry module*
 - *anyblok.migration module*
 - *anyblok.field module*
 - *anyblok.column module*
 - *anyblok.relationship module*
 - *anyblok._graphviz module*
 - *anyblok.scripts module*

anyblok module

`anyblok.start` (*processName*, *configuration_groups=None*, *entry_points=None*, *useseparator=False*,
loadwithoutmigration=False, *config=None*, ***kwargs*)
Function which initialize the application

```
registry = start('My application',  
                 configuration_groups=['config', 'database'],  
                 entry_points=['AnyBlok'])
```

Parameters

- **processName** – Name of the application
- **version** – Version of the application
- **prompt** – Prompt message for the help
- **configuration_groups** – list of the group of option for argparse
- **entry_points** – entry point where load blok
- **useseparator** – boolean, indicate if configuration option are split between two application
- **withoutautomigration** – if True, any

Return type registry if the database name is in the configuration

anyblok.declarations module

exception `anyblok.declarations.DeclarationsException`
Bases: `AttributeError`

Simple Exception for Declarations

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class anyblok.declarations.Declarations

Represents all the declarations done by the bloks

Warning: This is a global information, during the execution you must use the registry. The registry is the real assembler of the python classes based on the installed bloks

```
from anyblok import Declarations
```

class AuthorizationBinding

Encodes which policy to use per model or (model, permission).

In the assembly phase, copies of the policy are issued, and the registry is set as an attribute on them. This is a bit memory inefficient, but otherwise, passing the registry would have to be in all AuthorizationRule API calls.

class Declarations.Core

The Core class is the base of all the AnyBlok models

Add new core model:

```
@Declarations.register(Declarations.Core)
class Base:
    pass
```

Remove the core model:

```
Declarations.unregister(Declarations.Core, 'Base', Base,
                        blok='MyBlok')
```

classmethod register (parent, name, cls_, **kwargs)

Add new sub registry in the registry

Parameters

- **parent** – Existing declaration
- **name** – Name of the new declaration to add it
- **cls** – Class Interface to add in the declaration

classmethod unregister (entry, cls_)

Remove the Interface from the registry

Parameters

- **entry** – entry declaration of the model where the cls_ must be removed
- **cls** – Class Interface to remove in the declaration

class Declarations.Mixin

The Mixin class are used to define a behaviours on models:

- Add new mixin class:

```
@Declarations.register(Declarations.Mixin)
class MyMixinclass:
    pass
```

- Remove a mixin class:

```
Declarations.unregister(Declarations.Mixin.MyMixinclass, MyMixinclass)
```

class Declarations.**Model**

The Model class is used to define or inherit an SQL table.

Add new model class:

```
@Declarations.register(Declarations.Model)
class MyModelclass:
    pass
```

Remove a model class:

```
Declarations.unregister(Declarations.Model.MyModelclass,
                        MyModelclass)
```

There are three Model families:

- No SQL Model: These models have got any field, so any table
- SQL Model:
- SQL View Model: it is a model mapped with a SQL View, the insert, update delete method are forbidden by the database

Each model has a:

- registry name: compose by the parent + . + class model name
- table name: compose by the parent + ‘_’ + class model name

The table name can be overloaded by the attribute tablename. the wanted value are a string (name of the table) of a model in the declaration.

..warning:

```
Two models can have the same table name, both models are mapped on
the table. But they must have the same column.
```

classmethod **apply_event_listener** (*attr, method, registry, namespace, base, properties*)

Find the event listener methods in the base to save the namespace and the method in the registry

Parameters

- **attr** – name of the attribute
- **method** – method pointer
- **registry** – the current registry
- **namespace** – the namespace of the model
- **base** – One of the base of the model
- **properties** – the properties of the model

classmethod **apply_hybrid_method** (*base, registry, namespace, bases, transformation_properties, properties*)

Create overload to define the write declaration of sqlalchemy hybrid method, add the overload in the declared bases of the namespace

Parameters

- **registry** – the current registry
- **namespace** – the namespace of the model
- **base** – One of the base of the model
- **transformation_properties** – the properties of the model
- **properties** – assembled attributes of the namespace

classmethod `apply_sqlalchemy_event_listener` (*attr, method, registry, namespace, base, properties*)

declare in the registry the sqlalchemy event

Parameters

- **attr** – name of the attribute
- **method** – method pointer
- **registry** – the current registry
- **namespace** – the namespace of the model
- **base** – One of the base of the model
- **properties** – the properties of the model

classmethod `apply_table_and_mapper_args` (*base, registry, namespace, bases, transformation_properties, properties*)

Create overwrite to define table and mapper args to define some options for SQLAlchemy

Parameters

- **registry** – the current registry
- **namespace** – the namespace of the model
- **base** – One of the base of the model
- **transformation_properties** – the properties of the model
- **properties** – assembled attributes of the namespace

classmethod `apply_view` (*namespace, tablename, base, registry, properties*)

Transform the sqlalchemy model to view model

Parameters

- **namespace** – Namespace of the model
- **tablename** – Name of the table of the model
- **base** – Model cls
- **registry** – current registry
- **properties** – properties of the model

Exception MigrationException

Exception ViewException

classmethod `assemble_callback` (*registry*)

Assemble callback is called to assemble all the Model from the installed blocks

Parameters **registry** – registry to update

classmethod `declare_field` (*registry, name, field, namespace, properties*)

Declare the field/column/relationship to put in the properties of the model

Parameters

- **registry** – the current registry
- **name** – name of the field / column or relationship
- **field** – the declaration field / column or relationship
- **namespace** – the namespace of the model
- **properties** – the properties of the model

classmethod `detect_hybrid_method` (*attr, method, registry, namespace, base, properties*)

Find the sqlalchemy hybrid methods in the base to save the namespace and the method in the registry

Parameters

- **attr** – name of the attribute
- **method** – method pointer
- **registry** – the current registry
- **namespace** – the namespace of the model
- **base** – One of the base of the model
- **properties** – the properties of the model

classmethod `detect_table_and_mapper_args` (*registry, namespace, base, properties*)

Test if define_table/mapper_args are in the base, and call them save the value in the properties

if `__table/mapper_args__` are in the base then raise `ModelException`

Parameters

- **registry** – the current registry
- **namespace** – the namespace of the model
- **base** – One of the base of the model
- **properties** – the properties of the model

Exception `ModelException`

classmethod `initialize_callback` (*registry*)

initialize callback is called after assembling all entries

This callback updates the database information about

- `Model`
- `Column`
- `RelationShip`

Parameters **registry** – registry to update

classmethod `insert_in_bases` (*registry, namespace, bases, transformation_properties, properties*)

Add in the declared namespaces new base.

Parameters

- **registry** – the current registry
- **namespace** – the namespace of the model
- **base** – One of the base of the model
- **transformation_properties** – the properties of the model
- **properties** – assembled attributes of the namespace

classmethod `load_namespace_first_step` (*registry, namespace*)

Return the properties of the declared bases for a namespace. This is the first step because some actions need to know all the properties

Parameters

- **registry** – the current registry
- **namespace** – the namespace of the model

Return type dict of the known properties

classmethod `load_namespace_second_step` (*registry, namespace, realregistryname=None, transformation_properties=None*)

Return the bases and the properties of the namespace

Parameters

- **registry** – the current registry
- **namespace** – the namespace of the model
- **realregistryname** – the name of the model if the namespace is a mixin

Return type the list of the bases and the properties

Exception `ModelException`

classmethod `register` (*parent, name, cls_, **kwargs*)

add new sub registry in the registry

Parameters

- **parent** – Existing global registry
- **name** – Name of the new registry to add it
- **cls** – Class Interface to add in registry

classmethod `transform_base` (*registry, namespace, base, properties*)

Detect specific declaration which must define by registry

Parameters

- **registry** – the current registry
- **namespace** – the namespace of the model

- **base** – One of the base of the model
- **properties** – the properties of the model

Return type new base

classmethod `unregister(entry, cls_)`

Remove the Interface from the registry

Parameters

- **entry** – entry declaration of the model where the `cls_` must be removed
- **cls** – Class Interface to remove in registry

classmethod `Declarations.add_declaration_type(cls_=None, isAnEntry=False, as-
semble=None, initialize=None, un-
load=None)`

Add a declaration type

Parameters

- **cls** – The `class` object to add as a world of the `MetaData`
- **isAnEntry** – if true the type will be assembled by the registry
- **assemble** – name of the method callback to call (classmethod)
- **initialize** – name of the method callback to call (classmethod)
- **unload** – name of the method callback to call (classmethod)

Exception `DeclarationsException`

classmethod `Declarations.register(parent, cls_=None, **kwargs)`

Method to add the blok in the registry under a type of declaration

Parameters

- **parent** – An existing blok class in the Declaration
- **cls_** – The `class` object to add in the Declaration

Return type `cls_`

Exception `DeclarationsException`

classmethod `Declarations.unregister(entry, cls_)`

Method to remove the blok from a type of declaration

Parameters

- **entry** – declaration entry of the model where the `cls_` must be removed
- **cls_** – The `class` object to remove from the Declaration

Return type `cls_`

anyblok.model module

exception `anyblok.model.ModelException`

Bases: `Exception`

Exception for Model declaration

with_traceback()

`Exception.with_traceback(tb)` – set `self.__traceback__` to `tb` and return `self`.

exception `anyblok.model.ViewException`

Bases: `anyblok.model.ModelException`

Exception for View declaration

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

anyblok.mapper module

exception `anyblok.mapper.ModelAttributeException`

Bases: `Exception`

Exception for Model attribute

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `anyblok.mapper.ModelReprException`

Bases: `Exception`

Exception for Model attribute

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `anyblok.mapper.ModelAttributeAdapterException`

Bases: `Exception`

Exception for Model attribute adapter

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `anyblok.mapper.MapperException`

Bases: `AttributeError`

Simple Exception for Mapper

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `anyblok.mapper.ModelRepr(model_name)`

Pseudo class to represent a model

```
mr = ModelRepr('registry name')
```

check_model (registry)

Check if the model exist else raise an exception

Parameters **registry** – instance of the registry

Return type dict which represent the first step of the model

Exceptions `ModelReprException`

foreign_keys_for (registry, remote_model)

Return the of the primary keys

Parameters **registry** – instance of the registry

Return type list of `ModelAttribute`

primary_keys (*registry*)

Return the of the primary keys

Parameters **registry** – instance of the registry

Return type list of ModelAttribute

tablename (*registry*)

Return the real tablename of the Model

Parameters **registry** – instance of the registry

Return type string

class anyblok.mapper.**ModelAttribute** (*model_name, attribute_name*)

The Model attribute represente the using of a declared attribute, in the goal of get the real attribute after of the foreign_key:

```
ma = ModelAttribute('registry name', 'attribute name')
```

get_attribute (*registry, usehybrid=True*)

Return the assembled attribute, the model need to be assembled

Parameters

- **registry** – instance of the registry
- **usehybrid** – if True return the hybrid property if exist

Return type instance of the attribute

Exceptions ModelAttributeException

get_column_name (*registry*)

Return the name of the column

the need of foreign key may be before the creation of the model in the registry, so we must use the first step of assembling

Parameters **registry** – instance of the registry

Return type str of the foreign key (tablename.columnname)

Exceptions ModelAttributeException

get_fk (*registry*)

Return the foreign key which represent the attribute in the data base

Parameters **registry** – instance of the sqlalchemy ForeignKey

Return type instance of the attribute

get_fk_column (*registry*)

Return the foreign key which represent the attribute in the data base

Parameters **registry** – instance of the sqlalchemy ForeignKey

Return type instance of the attribute

get_fk_mapper (*registry*)

Return the foreign key which represent the attribute in the data base

Parameters **registry** – instance of the sqlalchemy ForeignKey

Return type instance of the attribute

get_fk_name (*registry*)

Return the name of the foreign key

the need of foreign key may be before the creation of the model in the registry, so we must use the first step of assembling

Parameters **registry** – instance of the registry

Return type str of the foreign key (tablename.columnname)

Exceptions ModelAttributeException

options (***kwargs*)

Add foreign key options to create the sqlalchemy ForeignKey

Parameters ****kwargs** – options

Return type the instance of the ModelAttribute

class anyblok.mapper.**ModelMapper** (*mapper, event, *args, **kwargs*)

class anyblok.mapper.**ModelAttributeMapper** (*mapper, event, *args, **kwargs*)

anyblok.mapper.**ModelAttributeAdapter** (*Model*)

Return a ModelAttribute

Parameters **Model** – ModelAttribute or string ('registry name'=>'attribute name')

Return type instance of ModelAttribute

Exceptions ModelAttributeAdapterException

anyblok.mapper.**ModelAdapter** (*Model*)

Return a ModelRepr

Parameters **Model** – ModelRepr or string

Return type instance of ModelRepr

Exceptions ModelAdapterException

anyblok.mapper.**MapperAdapter** (*mapper, *args, **kwargs*)

anyblok.config module

exception anyblok.config.**ConfigurationException**

Bases: LookupError

Simple Exception for Configuration

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class anyblok.config.**Configuration**

Configuration is used to define the options of the real argparse and its default values. Each application or blok can declare needed options here.

This class stores three attributes:

- groups: lists of options indexed by part, a part is a ConfigParser group, or a process name
- labels: if a group has got a label then all the options in group are gathered in a parser group
- configuration: result of the Configuration after loading

classmethod add (*group*, *part*='blocs', *label*=None, *function_*=None,
must_be_loaded_by_unittest=False)

Add a function in a part and a group.

The function must have two arguments:

- **parser**: the parser instance of argparse
- **default**: A dict with the default value

This function is called to know what the options of this must do. You can declare this group:

- either by calling the add method as a function:

```
def foo(parser, default):  
    pass  
  
Configuration.add('create-db', function_=foo)
```

- or by calling the add method as a decorator:

```
@Configuration.add('create-db')  
def bar(parser, default):  
    pass
```

By default the group is unnamed, if you want a named group, you must set the `label` attribute:

```
@Configuration.add('create-db', label="Name of the group")  
def bar(parser, default):  
    pass
```

Parameters

- **part** – ConfigParser group or process name
- **group** – group is a set of parser option
- **label** – If the group has a label then all the functions in the group are put in group parser
- **function** – function to add
- **must_be_loaded_by_unittest** – unittest call this function to init configuration of AnyBlok for run unittest”

classmethod get (*opt*, *default*=None)

Get a value from the configuration dict after loading

After the loading of the application, all the options are saved in the Configuration. And all the applications have free access to these options:

```
from anyblok._configuration import Configuration  
  
database = Configuration.get('db_name')
```

..warning:

```
Some options are used as a default value not real value, such  
as the db_name
```

Parameters

- **opt** – name of the option

- **default** – default value if the option doesn't exist

classmethod has (*option*)

Check if the option exist in the configuration dict

Return True if the option is in the configuration dict and the value is not None. A None value is different than a ConfigOption with None value

Parameters **opt** – option key to check

Return type boolean True is exist

classmethod load (*application, configuration_groups=None, parts_to_load=('bloks',), useseparator=False, **kwargs*)

Load the argparse definition and parse them

Parameters

- **application** – name of the application
- **configuration_groups** – iterable configuration group to load
- **parts_to_load** – group of blok to load
- **useseparator** – boolean(default False)
- ****kwargs** – ArgumentParser named arguments

classmethod load_config_for_test ()

Load the argparse configuration need for the unittest

classmethod remove (*group,function_, part='bloks'*)

Remove an existing function

If your application inherits some unwanted options from a specific function, you can unlink this function:

```
def foo(opt, default):
    pass

Configuration.add('create-db', function_=foo)
Configuration.remove('create-db', function_=foo)
```

Parameters

- **part** – ConfigParser group or process name
- **group** – group is a set of parser option
- **function** – function to add

classmethod remove_label (*group, part='bloks'*)

Remove an existing label

The goal of this function is to remove an existing label of a specific group:

```
@Configuration.add('create-db', label="Name of the group")
def bar(parser, default):
    pass

Configuration.remove_label('create-db')
```

Parameters

- **part** – ConfigParser group or process name

- **group** – group is a set of parser option

classmethod **set** (*opt, value*)

Set a value to the configuration dict

Parameters

- **opt** – name of the option
- **value** – value to set

anyblok.logging module

class `anyblok.logging.consoleFormatter` (*fmt=None, datefmt=None, style='%'*)

Bases: `logging.Formatter`

Define the format for console logging

converter ()

localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When ‘seconds’ is not passed in, convert the current time instead.

format (*record*)

Add color to the message

Parameters **record** – logging record instance

Return type logging record formatted

formatException (*ei*)

Format and return the specified exception information as a string.

This default implementation just uses `traceback.print_exception()`

formatStack (*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to `traceback.print_stack()`, but with the last trailing newline removed.

The base implementation just returns the value passed in.

formatTime (*record, datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from `format()` by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if `datefmt` (a string) is specified, it is used with `time.strftime()` to format the creation time of the record. Otherwise, the ISO8601 format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, `time.localtime()` is used; to change this for a particular formatter instance, set the ‘converter’ attribute to a function with the same signature as `time.localtime()` or `time.gmtime()`. To change it for all formatters, for example if you want all logging times to be shown in GMT, set the ‘converter’ attribute in the `Formatter` class.

usesTime ()

Check if the format uses the creation time of the record.

class `anyblok.logging.anyblokFormatter` (*fmt=None, datefmt=None, style='%'*)

Bases: `logging.Formatter`

Define the format for console logging

converter ()

localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When 'seconds' is not passed in, convert the current time instead.

format (*record*)

Add color to the message

Parameters **record** – logging record instance

Return type logging record formatted

formatException (*ei*)

Format and return the specified exception information as a string.

This default implementation just uses `traceback.print_exception()`

formatStack (*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to `traceback.print_stack()`, but with the last trailing newline removed.

The base implementation just returns the value passed in.

formatTime (*record, datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from `format()` by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if `datefmt` (a string) is specified, it is used with `time.strftime()` to format the creation time of the record. Otherwise, the ISO8601 format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, `time.localtime()` is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as `time.localtime()` or `time.gmtime()`. To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the `Formatter` class.

usesTime ()

Check if the format uses the creation time of the record.

`anyblok.logging.log` (*logger, level='info', withargs=False*)

decorator to log the entry of a method

There are 5 levels of logging * debug * info (default) * warning * error * critical

example:

```
from logging import getLogger
logger = getLogger(__name__)

@log(logger)
def foo(...):
    ...
```

Parameters

- **level** – AnyBlok log level
- **withargs** – If True, add args and kwargs in the log message

anyblok.imp module

exception `anyblok.imp.ImportManagerException`

Bases: `AttributeError`

Exception for Import Manager

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `anyblok.imp.ImportManager`

Use to import the blok or reload the blok imports

Add a blok and imports its modules:

```
blok = ImportManager.add('my blok')
blok.imports()
```

Reload the modules of a blok:

```
if ImportManager.has('my blok'):
    blok = ImportManager.get('my blok')
    blok.reload()
    # import the unimported module
```

classmethod `add(blok)`

Store the blok so that we know which bloks to reload if needed

Parameters **blok** – name of the blok to add

Return type loader instance

Exception `ImportManagerException`

classmethod `get(blok)`

Return the module imported for this blok

Parameters **blok** – name of the blok to add

Return type loader instance

Exception `ImportManagerException`

classmethod `has(blok)`

Return True if the blok was imported

Parameters **blok** – name of the blok to add

Return type boolean

anyblok.environment module

exception `anyblok.environment.EnvironmentException`

Bases: `AttributeError`

Exception for the Environment


```

with_traceback ()
    Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class anyblok.environment.EnvironmentManager
    Manage the Environment for an application

    classmethod define_environment_cls (Environment)
        Define the class used for the environment

        Parameters Environment – class of environment

        Exception EnvironmentException

    environment
        alias of ThreadEnvironment

    classmethod get (key, default=None)
        Load the value of the key in the environment

        Parameters

        • key – the key of the value to load

        • default – return this value if not value loaded for the key

        Return type the value of the key

        Exception EnvironmentException

    classmethod scoped_function_for_session ()
        Save the value of the key in the environment

    classmethod set (key, value)
        Save the value of the key in the environment

        Parameters

        • key – the key of the value to save

        • value – the value to save

        Exception EnvironmentException

class anyblok.environment.ThreadEnvironment
    Use the thread, to get the environment

    classmethod getter (key, default)
        Get the value of the key in the environment

        Parameters

        • key – the key of the value to retrieve

        • default – return this value if no value loaded for the key

        Return type the value of the key

    scoped_function_for_session = None
        No scoped function here because for none value sqlalchemy already uses a thread to save the session

    classmethod setter (key, value)
        Save the value of the key in the environment

        Parameters

        • key – the key of the value to save

        • value – the value to save

```

anyblok.blok module

exception `anyblok.blok.BlokManagerException(*args, **kwargs)`

Bases: `LookupError`

Simple exception to `BlokManager`

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `anyblok.blok.BlokManager`

Manage the bloks for one process

A blok has a *setuptools* entrypoint, this entry point is defined by the `entry_points` attribute in the first load

The `bloks` attribute is a dict with all the loaded entry points

Use this class to import all the bloks in the entrypoint:

`BlokManager.load()`

classmethod `add_importer(key, cls_name)`

Add a new importer

Parameters

- **key** – key of the importer
- **cls_name** – name of the model to import

classmethod `get(blok)`

Return the loaded blok

Parameters **blok** – blok name

Return type blok instance

Exception `BlokManagerException`

classmethod `getPath(blok)`

Return the path of the blok

Parameters **blok** – blok name in `ordered_bloks`

Return type absolute path

classmethod `get_importer(key)`

Get the importer class name

Parameters **key** – key of the importer

Return type name of the model to import

Exception `BlokManagerException`

classmethod `has(blok)`

Return True if the blok is loaded

Parameters **blok** – blok name

Return type bool

classmethod `has_importer(key)`

Check if an importer

classmethod list()

Return the ordered bloks

Return type list of blok name ordered by loading

classmethod load(*entry_points*=(*'bloks'* ,))

Load all the bloks and import them

Parameters *entry_points* – Use by *iter_entry_points* to get the blok

Exception BlokManagerException

classmethod reload()

Reload the entry points

Empty the *bloks* dict and use the *entry_points* attribute to load bloks :exception: BlokManagerException

classmethod set(*blokname*, *blok*)

Add a new blok

Parameters

- **blokname** – blok name
- **blok** – blok instance

Exception BlokManagerException

classmethod unload()

Unload all the bloks but not the registry

class *anyblok.blok.Blok*(*registry*)

Super class for all the bloks

define the default value for:

- **priority**: order to load the blok
- **required**: list of the bloks needed to install this blok
- **optional**: list of the bloks to be installed if present in the blok list
- **conditional**: if all the bloks of this list are installed then install this blok

classmethod import_declaration_module()

Do the python import for the Declaration of the model or other

import_file(*importer_name*, *model*, **file_path*, ***kwargs*)

Import data file

Parameters

- **importer_name** – Name of the importer (need installation of the Blok which have the importer)
- **model** – Model of the data to import
- ***file_path** – relative path of the path in this Blok
- ****kwargs** – Option for the importer

Return type return dict of result

load()

Call at the launch of the application

post_migration (*latest_version*)

Call at update, after the automigration

Parameters **latest_version** – latest version installed, if the blok have not been installing the latest_version will be None

pre_migration (*latest_version*)

Call at update, before the automigration

Warning: You can not use the ORM

Parameters **latest_version** – latest version installed, if the blok have not been installing the latest_version will be None

uninstall ()

Call at the uninstallation

update (*latest_version*)

Call at the installation or update

Parameters **latest_version** – latest version installed, if the blok have not been installing the latest_version will be None

anyblok.registry module

exception anyblok.registry.**RegistryManagerException**

Bases: Exception

Simple Exception for Registry

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception anyblok.registry.**RegistryException**

Bases: Exception

Simple Exception for Registry

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class anyblok.registry.**RegistryManager**

Manage the global registry

Add new entry:

```
RegistryManager.declare_entry('newEntry')
RegistryManager.init_blok('newBlok')
EnvironmentManager.set('current_blok', 'newBlok')
RegistryManager.add_entry_in_register(
    'newEntry', 'oneKey', cls_)
EnvironmentManager.set('current_blok', None)
```

Remove an existing entry:

```
if RegistryManager.has_entry_in_register('newBlok', 'newEntry',
                                         'oneKey'):
```

```
RegistryManager.remove_entry_in_register(
    'newBlok', 'newEntry', 'oneKey', cls_)
```

get a new registry for a database:

```
registry = RegistryManager.get('my database')
```

classmethod add_core_in_register (*core*, *cls_*)

Load core in blok

warning the global var `current_blok` must be filled on the good blok

Parameters

- **core** – is the existing core name
- **cls_** – Class of the Core to save in loaded blok target registry

classmethod add_entry_in_register (*entry*, *key*, *cls_*, ***kwargs*)

Load entry in blok

warning the global var `current_blok` must be filled on the good blok :param *entry*: is the existing entry name :param *key*: is the existing key in the entry :param *cls_*: Class of the entry / key to remove in loaded blok

classmethod add_or_replace_blok_property (*property_*, *value*)

Save the value in the properties

Parameters

- **property_** – name of the property
- **value** – the value to save, the type is not important

classmethod clear ()

Clear the registry dict to force the creation of new registry

classmethod declare_core (*core*)

Add new core in the declared cores

```
RegistryManager.declare_core('Core name')

-----

@Declarations.register(Declarations.Core)
class ``Core name``:
    ...
```

Warning: The core must be declared in the application, not in the bloks The declaration must be done before the loading of the bloks

Parameters **core** – core name

classmethod declare_entry (*entry*, *assemble_callback=None*, *initialize_callback=None*)

Add new entry in the declared entries

```
def assemble_callback(registry):
    ...
```

```
def initialize_callback(registry):
    ...

RegistryManager.declare_entry(
    'Entry name', assemble_callback=assemble_callback,
    initialize_callback=initialize_callback)

@Declarations.register(Declarations.``Entry name``)
class MyClass:
    ...
```

Warning: The entry must be declared in the application, not in the bloks The declaration must be done before the loading of the bloks

Parameters

- **entry** – entry name
- **assemble_callback** – function callback to call to assemble
- **initialize_callback** – function callback to call to init after assembling

classmethod declare_unload_callback (*entry, unload_callback*)

Save a unload callback in registry Manager

Parameters

- **entry** – declaration type name
- **unload_callback** – classmethod pointer

classmethod get (*db_name, loadwithoutmigration=False, **kwargs*)

Return an existing Registry

If the Registry doesn't exist then the Registry are created and added to registries dict

Parameters db_name – the name of the database linked to this registry

Return type Registry

classmethod get_blok_property (*property_, default=None*)

Return the value in the properties

Parameters

- **property_** – name of the property
- **default** – return default If not entry in the property

classmethod has_blok (*blok*)

Return True if the blok is already loaded

Parameters blok – name of the blok

Return type boolean

classmethod has_blok_property (*property_*)

Return True if the property exists in blok

Parameters property_ – name of the property

classmethod has_core_in_register (*blok, core*)

Return True if One Class exist in this blok for this core

Parameters

- **blok** – name of the blok
- **core** – is the existing core name

classmethod `has_entry_in_register` (*blok, entry, key*)

Return True if One Class exist in this blok for this entry

Parameters

- **blok** – name of the blok
- **entry** – is the existing entry name
- **key** – is the existing key in the entry

classmethod `init_blok` (*blokname*)

init one blok to be known by the RegistryManager

All blos loaded must be initialized because the registry will be created with this information

Parameters **blokname** – name of the blok

classmethod `reload` ()

Reload the blok

The purpose is to reload the python module to get changes in python file

classmethod `remove_blok_property` (*property_*)

Remove the property if exist

Parameters **property_** – name of the property

classmethod `remove_in_register` (*cls_*)

Remove Class in blok and in entry

Parameters **cls_** – Class of the entry / key to remove in loaded blok

classmethod `unload` ()

Call all the unload callbacks

class `anyblok.registry.Registry` (*db_name, loadwithoutmigration=False, unittest=False, **kwargs*)

Define one registry

A registry is linked to a database, and stores the definition of the installed Blos, Models, Mixins for this database:

```
registry = Registry('My database')
```

add_in_registry (*namespace, base*)

Add a class as an attribute of the registry

Parameters

- **namespace** – tree path of the attribute
- **base** – class to add

apply_state (*blok_name, state, in_states*)

Apply the state of the blok name

Parameters

- **blok_name** – the name of the blok

- **state** – the state to apply
- **in_states** – the blok must be in this state

Exception RegistryException

check_permission (*target, principals, permission*)

Check that one of the principals has permisson on target.

Parameters

- **target** – model instance (record) or class. Checking a permission on a model class with a policy that needs to work on records is considered a configuration error: the policy has the right to fail.
- **principals** – list, set or tuple of strings

Return type bool

clean_model ()

Clean the registry of all the namespaces

close ()

Release the session, connection and engine

close_session ()

Close only the session, not the registry After the call of this method the registry won't be usable you should use close method which call this method

commit (*args, **kwargs)

Overload the commit method of the SQLAlchemy session

complete_reload ()

Reload the code and registry

create_session_factory ()

Create the SQLA Session factory

in function of the Core Session class ans the Core Qery class

engine

property to get the engine

expire (*obj, attribute_names=None*)

Expire object in session, you can define some attribute which are expired:

```
registry.expire(instance, ['attr1', 'attr2', ...])
```

Parameters

- **obj** – instance of Model
- **attribute_names** – list of string, names of the attr to expire

get (*namespace*)

Return the namespace Class

Parameters **namespace** – namespace to get from the registry str

Return type namespace cls

Exception RegistryManagerException

get_bloks_by_states (*states)

Return the bloks in these states

Parameters **states** – list of the states

Return type list of blok's name

get_bloks_to_install (*loaded*)

Return the bloks to install in the registry

Return type list of blok's name

get_bloks_to_load ()

Return the bloks to load by the registry

Return type list of blok's name

ini_var ()

Initialize the var to load the registry

init_bind ()

Initialize the bind

init_engine (*db_name=None*)

Define the engine

Parameters **db_name** – name of the database to link

init_engine_options ()

Define the options to initialize the engine

load ()

Load all the namespaces of the registry

Create all the table, make the shema migration Update Blok, Model, Column rows

load_blok (*blok, toinstall, toload*)

load on blok, load all the core and all the entry for one blok

Parameters **blok** – name of the blok

Exception RegistryManagerException

load_core (*blok, core*)

load one core type for one blok

Parameters

- **blok** – name of the blok
- **core** – the core name to load

load_entry (*blok, entry*)

load one entry type for one blok

Parameters

- **blok** – name of the blok
- **entry** – declaration type to load

lookup_policy (*target, permission*)

Return the policy instance that applies to target or its model.

Parameters **target** – model class or instance

If a policy is declared for the precise permission, it is returned. Otherwise, the default policy for that model is returned. By ultimate default the special `anyblok.authorization.rule.DenyAll` is returned.

must_recreate_session_factory()

Check if the SQLA Session Factory must be destroy and recreate

Return type Boolean, True if nb Core Session/Query inheritance change

precommit_hook (*registryname, method, *args, **kwargs*)

Add a method in the precommit_hook list

a precommit hook is a method called just after the commit, it is used to call this method once, because a hook is saved only once

Parameters

- **registryname** – namespace of the model
- **method** – method to call on the registryname
- **put_at_the_end_if_exist** – if true and hook already exist then the hook are moved at the end

refresh (*obj, attribute_names=None*)

Expire and reload object in session, you can define some attribute which are refreshed:

```
registry.refresh(instance, ['attr1', 'attr2', ...])
```

Parameters

- **obj** – instance of Model
- **attribute_names** – list of string, names of the attr to refresh

reload()

Reload the registry, close session, clean registry, reinit var

upgrade (*install=None, update=None, uninstall=None*)

Upgrade the current registry

Parameters

- **install** – list of the blok to install
- **update** – list of the blok to update
- **uninstall** – list of the blok to uninstall

Exception RegistryException

wrap_query_permission (*query, principals, permission, models=()*)

Wrap query to return only authorized results

Parameters

- **principals** – list, set or tuple of strings
- **models** – models on which to apply security filtering. If not supplied, it will be inferred from the query. The length and ordering much match that of expected results.

Returns a query-like object, implementing the results fetching API, but that can't be further filtered.

This method calls all the relevant policies to apply pre- and post-filtering. Although postfiltering is discouraged in authorization policies for performance and expressiveness (limit, offset), there are cases for which it is unavoidable, or in which the tradeoff goes the other way.

In normal operation, the relevant models are inferred directly from the query. For join situations, and more complex queries, the caller has control on the models on which to exert permission checking.

For instance, it might make sense to use a join between Model1 and Model2 to actually constrain Model1 (on which permission filtering should occur) by information contained in Model2, even if the passed principals should not grant access to the relevant Model2 records.

anyblok.migration module

Warning: AnyBlok use Alembic to do the dynamic migration, but Alembic does'nt detect all the change (primary key, ...), we must wait the Alembic or implement it in Alembic project before use it in AnyBlok

exception anyblok.migration.**MigrationException**

Bases: `AttributeError`

Simple Exception class for Migration

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class anyblok.migration.**MigrationReport** (migration, diffs)

Change report

Get a new report:

```
report = MigrationReport(migrationinstance, change_detected)
```

apply_change ()

Apply the migration

this method parses the detected change and calls the Migration system to apply the change with the api of Declarations

log_has (log)

return True id the log is present

Warning: this method is only used for the unittest

Parameters **log** – log sentence expected

class anyblok.migration.**MigrationConstraintForeignKey** (table, name)

Used to apply a migration on a foreign key

You can add:

```
table.column('my column').foreign_key().add(Blok.name)
```

Or drop:

```
table.column('my column').foreign_key().drop()
```

add (local_columns, remote_columns, **kwargs)

Add a new foreign key

Parameters **remote_field** – The column of the remote model

Return type MigrationConstraintForeignKey instance

drop()

Drop the foreign key

class anyblok.migration.**MigrationColumn**(table, name)

get or add a column

Add a new column:

```
table.column().add(Sqlalchemy column)
```

Get a column:

```
c = table.column('My column name')
```

Alter the column:

```
c.alter(new_column_name='Another column name')
```

Drop the column:

```
c.drop()
```

add(column)

Add a new column

The column is added in two phases, the last phase is only for the nullable, if nullable can not be applied, a warning is logged

Parameters **column** – sqlalchemy column

Return type MigrationColumn instance

alter(**kwargs)

Alter an existing column

Alter the column in two phases, because the nullable column has not locked the migration

Warning: See Alembic alter_column, the existing_* param are used for some dialect like mysql, is importante to filled them for these dialect

Parameters

- **new_column_name** – New name for the column
- **type** – New sqlalchemy type
- **existing_type** – Old sqlalchemy type
- **server_default** – The default value in database server
- **existing_server_default** – Old default value
- **nullable** – New nullable value
- **existing_nullable** – Old nullable value
- **autoincrement** – New auto increment use for Integer whith primary key only
- **existing_autoincrement** – Old auto increment

Return type MigrationColumn instance

drop()

Drop the column

nullable()

Use for unittest return if the column is nullable

server_default()

Use for unittest: return the default database value

type()

Use for unittest: return the column type

class anyblok.migration.**MigrationConstraintCheck**(*table, name*)

Used for the Check constraint

Add a new constraint:

```
table('My table name').check().add('check_my_column', 'mycolumn > 5')
```

Get and drop the constraint:

```
table('My table name').check('check_my_column').drop()
```

add(*condition*)

Add the constraint

Parameters **condition** – constraint to apply

Return type MigrationConstraintCheck instance

drop()

Drop the constraint

class anyblok.migration.**MigrationConstraintUnique**(*table, name*)

Used for the Unique constraint

Add a new constraint:

```
table('My table name').unique('constraint name').add('col1', 'col2')
```

Get and drop the constraint:

```
table('My table name').unique('constraint name').drop()
```

Let AnyBlok to define the name of the constraint:

```
table('My table name').unique(None).add('col1', 'col2')
```

add(**columns*)

Add the constraint

Parameters ***column** – list of column name

Return type MigrationConstraintUnique instance

Exception MigrationException

drop()

Drop the constraint

class anyblok.migration.**MigrationConstraintPrimaryKey**(*table*)

Used for the primary key constraint

Add a new constraint:

```
table('My table name').primarykey().add('col1', 'col2')
```

Get and drop the constraint:

```
table('My table name').primarykey('col1', 'col2').drop()
```

add (*columns)

Add the constraint

Parameters *column – list of column name

Return type MigrationConstraintPrimaryKey instance

Exception MigrationException

drop ()

Drop the constraint

class anyblok.migration.**MigrationIndex** (table, *columns, **kwargs)

Used for the index constraint

Add a new constraint:

```
table('My table name').index().add('col1', 'col2')
```

Get and drop the constraint:

```
table('My table name').index('col1', 'col2').drop()
```

add (*columns)

Add the constraint

Parameters *column – list of column name

Return type MigrationIndex instance

Exception MigrationException

drop ()

Drop the constraint

class anyblok.migration.**MigrationTable** (migration, name)

Use to manipulate tables

Add a table:

```
table().add('New table')
```

Get an existing table:

```
t = table('My table name')
```

Alter the table:

```
t.alter(name='Another table name')
```

Drop the table:

```
t.drop()
```

add (*name*)

Add a new table

Parameters *name* – name of the table

Return type MigrationTable instance

alter (***kwargs*)

Alter the current table

Parameters *name* – New table name

Return type MigrationTable instance

Exception MigrationException

check (*name=None*)

Get check

Parameters **columns* – List of the column's name

Return type MigrationConstraintCheck instance

column (*name=None*)

Get Column

Parameters *name* – Column name

Return type MigrationColumn instance

drop ()

Drop the table

foreign_key (*name*)

Get a foreign key

Return type MigrationConstraintForeignKey instance

index (**columns, **kwargs*)

Get index

Parameters **columns* – List of the column's name

Return type MigrationIndex instance

primarykey ()

Get primary key

Parameters **columns* – List of the column's name

Return type MigrationConstraintPrimaryKey instance

unique (*name*)

Get unique

Parameters **columns* – List of the column's name

Return type MigrationConstraintUnique instance

class anyblok.migration.**Migration** (*registry*)

Migration Main entry

This class allows to manipulate all the migration class:

```
migration = Migration(Session(), Base.Metadata)
t = migration.table('My table name')
c = t.column('My column name from t')
```

auto_upgrade_database ()

Upgrade the database automatically

detect_changed ()

Detect the difference between the metadata and the database

Return type MigrationReport instance

release_savepoint (*name*)

Release the save point

Parameters **name** – name of the savepoint

rollback_savepoint (*name*)

Rollback to the savepoint

Parameters **name** – name of the savepoint

savepoint (*name=None*)

Add a savepoint

Parameters **name** – name of the save point

Return type return the name of the save point

table (*name=None*)

Get a table

Return type MigrationTable instance

anyblok.field module

class anyblok.field.**Field** (*args, **kwargs)

Field class

This class must not be instantiated

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry

- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known of the model

Return type instance of Field

must_be_declared_as_attr ()

Return False, it is the default value

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

Exception FieldException

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.field.**Function** (**args, **kwargs*)

Bases: anyblok.field.Field

Function Field

```
from anyblok.declarations import Declarations
from anyblok.field import Function

@Declarations.register(Declarations.Model)
class Test:
    x = Function(fget='fget', fset='fset', fdel='fdel', fexpr='fexpr')
```

```
..warning::  
  
fexp must be a classmethod
```

forbid_instance (*cls*)

Raise an exception if the *cls* is an instance of this `__class__`

Parameters *cls* – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters *fieldname* – if no label filled, the *fieldname* will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the `get` is called in `classattribute`, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the property of the field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

must_be_declared_as_attr ()

Return False, it is the default value

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

Exception `FieldException`

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

anyblok.column module

class `anyblok.column.Column` (*args, **kwargs)

Bases: `anyblok.field.Field`

Column class

This class can't be instantiated

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this `__class__`

Parameters **cls** – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added()

Return False, it is the default value

native_type()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**Integer** (**args, **kwargs*)

Bases: anyblok.column.Column

Integer column

```
from anyblok.declarations import Declarations
from anyblok.column import Integer

@Declarations.register(Declarations.Model)
class Test:

    x = Integer(default=1)
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

sqlalchemy_type

alias of Integer

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)
Return a default getter for the field

Parameters **fieldname** – name of the field

class `anyblok.column.SmallInteger` (**args, **kwargs*)

Bases: `anyblok.column.Column`

Small integer column

```
from anyblok.declarations import Declarations
from anyblok.column import SmallInteger

@Declarations.register(Declarations.Model)
class Test:

    x = SmallInteger(default=1)
```

forbid_instance (*cls*)

Raise an exception if the `cls` is an instance of this `__class__`

Parameters **cls** – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the `fieldname` will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the `get` is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr()
Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added()
Return False, it is the default value

native_type()
Return the native SQLAlchemy type

sqlalchemy_type
alias of `SmallInteger`

update_properties (*registry, namespace, fieldname, properties*)
Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)
Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)
Return a default getter for the field

Parameters **fieldname** – name of the field

class `anyblok.column.BigInteger` (**args, **kwargs*)
Bases: `anyblok.column.Column`
Big integer column

```
from anyblok.declarations import Declarations
from anyblok.column import BigInteger

@Declarations.register(Declarations.Model)
class Test:

    x = BigInteger(default=1)
```

forbid_instance (*cls*)
Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception `FieldException`

format_label (*fieldname*)
Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)
Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

sqlalchemy_type

alias of BigInteger

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**Boolean** (*args, **kwargs)

Bases: anyblok.column.Column

Boolean column

```
from anyblok.declarations import Declarations
from anyblok.column import Boolean

@Declarations.register(Declarations.Model)
class Test:

    x = Boolean(default=True)
```

forbid_instance (cls)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (fieldname)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (registry, namespace, fieldname, properties)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (registry, namespace, fieldname, properties)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added()

Return False, it is the default value

native_type()

Return the native SQLAlchemy type

sqlalchemy_type

alias of Boolean

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class `anyblok.column.Float` (**args, **kwargs*)

Bases: `anyblok.column.Column`

Float column

```
from anyblok.declarations import Declarations
from anyblok.column import Float

@Declarations.register(Declarations.Model)
class Test:

    x = Float(default=1.0)
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

sqlalchemy_type

alias of Float

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class `anyblok.column.Decimal(*args, **kwargs)`
Bases: `anyblok.column.Column`

Decimal column

```
from decimal import Decimal as D
from anyblok.declarations import Declarations
from anyblok.column import Decimal

@Declarations.register(Declarations.Model)
class Test:

    x = Decimal(default=D('1.1'))
```

forbid_instance (*cls*)

Raise an exception if the *cls* is an instance of this `__class__`

Parameters *cls* – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters *fieldname* – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the `get` is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added()

Return False, it is the default value

native_type()

Return the native SQLAlchemy type

sqlalchemy_type

alias of DECIMAL

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class `anyblok.column.Date(*args, **kwargs)`

Bases: `anyblok.column.Column`

Date column

```
from datetime import date
from anyblok.declarations import Declarations
from anyblok.column import Date

@Declarations.register(Declarations.Model)
class Test:

    x = Date(default=date.today())
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

sqlalchemy_type

alias of Date

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

```
class anyblok.column.DateTime(*args, **kwargs)
    Bases: anyblok.column.Column
```

DateTime column

```
from datetime import datetime
from anyblok.declarations import Declarations
from anyblok.column import DateTime

@Declarations.register(Declarations.Model)
class Test:

    x = DateTime(default=datetime.now)
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added()

Return False, it is the default value

native_type()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class `anyblok.column.Time` (**args, **kwargs*)

Bases: `anyblok.column.Column`

Time column

```
from datetime import time
from anyblok.declarations import Declarations
from anyblok.column import Time

@Declarations.register(Declarations.Model)
class Test:

    x = Time(default=time())
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

sqlalchemy_type

alias of Time

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**Interval** (**args, **kwargs*)

Bases: anyblok.column.Column

Datetime interval column

```
from datetime import timedelta
from anyblok.declarations import Declarations
from anyblok.column import Interval

@Declarations.register(Declarations.Model)
class Test:

    x = Interval(default=timedelta(days=5))
```

forbid_instance (*cls*)

Raise an exception if the *cls* is an instance of this `__class__`

Parameters *cls* – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters *fieldname* – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the `get` is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

sqlalchemy_type

alias of Interval

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**String** (**args, **kwargs*)

Bases: anyblok.column.Column

String column

```
from anyblok.declarations import Declarations
from anyblok.column import String

@Declarations.register(Declarations.Model)
class Test:

    x = String(default='test')
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry

- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.uString (**args, **kwargs*)

Bases: anyblok.column.Column

Unicode column

```
from anyblok.declarations import Declarations
from anyblok.column import uString

@Declarations.register(Declarations.Model)
class Test:

    x = uString(de", default=u'test')
```

forbid_instance (*cls*)

Raise an exception if the *cls* is an instance of this `__class__`

Parameters *cls* – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters *fieldname* – if no label filled, the *fieldname* will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the `get` is called in `classattribute`, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**Text** (*args, **kwargs)

Bases: anyblok.column.Column

Text column

```
from anyblok.declarations import Declarations
from anyblok.column import Text

@Declarations.register(Declarations.Model)
class Test:

    x = Text(default='test')
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry

- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**uText** (*args, **kwargs)

Bases: anyblok.column.Column

Unicode text column

```
from anyblok.declarations import Declarations
from anyblok.column import uText

@Declarations.register(Declarations.Model)
class Test:

    x = uText(default=u'test')
```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class `anyblok.column.StrSelection`

Class representing the data of one column Selection

class `anyblok.column.SelectionType` (*selections, size, registry=None, namespace=None*)

Generic type for Column Selection

impl

alias of String

class `anyblok.column.Selection` (**args, **kwargs*)

Bases: `anyblok.column.Column`

Selection column

```
from anyblok.declarations import Declarations
from anyblok.column import Selection

@Declarations.register(Declarations.Model)
class Test:
    STATUS = (
        (u'draft', u'Draft'),
        (u'done', u'Done'),
    )

    x = Selection(selections=STATUS, size=64, default=u'draft')
```

forbid_instance (*cls*)

Raise an exception if the *cls* is an instance of this `__class__`

Parameters *cls* – instance of the class

Exception `FieldException`

format_label (*fieldname*)

Return the label for this field

Parameters *fieldname* – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the `get` is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return True because the field selection in a mixin must be copied else the selection method can be wrong

native_type()

Return the native SQLAlchemy type

update_properties(*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column(*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column(*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**JsonType**(**args, **kwargs*)

Generic type for Column JSON

impl

alias of Unicode

class anyblok.column.**Json**(**args, **kwargs*)

Bases: anyblok.column.Column

JSON column

```
from anyblok.declarations import Declarations
from anyblok.column import Json

@Declarations.register(Declarations.Model)
class Test:

    x = Json()
```

forbid_instance(*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label(*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property(*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

sqlalchemy_type

alias of JsonType

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**LargeBinary** (*args, **kwargs)
Bases: anyblok.column.Column

Large binary column

```
from os import urandom
from anyblok.declarations import Declarations
from anyblok.column import LargeBinary

blob = urandom(100000)

@Declarations.register(Declarations.Model)
class Test:

    x = LargeBinary(default=blob)
```

forbid_instance (cls)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (fieldname)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (registry, namespace, fieldname, properties)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (registry, namespace, fieldname, properties)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr()
Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added()
Return False, it is the default value

native_type()
Return the native SQLAlchemy type

sqlalchemy_type
alias of LargeBinary

update_properties (*registry, namespace, fieldname, properties*)
Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)
Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)
Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**Color** (**args, **kwargs*)
Bases: anyblok.column.Column
Color column. See [coulour](#) package

```
from anyblok.declarations import Declarations
from anyblok.column import Color

@Declarations.register(Declarations.Model)
class Test:

    x = Color(default='green')
```

forbid_instance (*cls*)
Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)
Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)
Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.column.**Password** (**args, **kwargs*)

Bases: anyblok.column.Column

String column

```

from anyblok.declarations import Declarations
from anyblok.column import Password

@Declarations.register(Declarations.Model)
class Test:

    x = Password(crypt_context={'schemes': ['md5_crypt']})

=====

test = Test.insert()
test.x = 'mypassword'

test.x
==> Password object with encrypt value, the value can not be read

test.x == 'mypassword'
==> True

```

..warning:

```

the column type Password can not be querying::

Test.query().filter(Test.x == 'mypassword').count()
==> 0

```

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – known properties of the model

Return type sqlalchemy column instance

must_be_declared_as_attr ()

Return True if the column have a foreign key to a remote column

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

anyblok.relationship module

class anyblok.relationship.**Relationship** (**args, **kwargs*)

Bases: anyblok.field.Field

Relationship class

The Relationship class is used to define the type of SQL field Declarations

Add a new relation ship type:

```
@Declarations.register(Declarations.Relationship)
class Many2one:
    pass
```

the relationship column are forbidden because the model can be used on the model

apply_instrumentedlist (*registry, namespace, fieldname*)

Add the InstrumentedList class to replace List class as result of the query

Parameters **registry** – current registry

define_backref_properties (*registry, namespace, properties*)

Add in the backref_properties, new property for the backref

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **properties** – properties known of the model

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_backref (*registry, namespace, fieldname, properties*)

Create the real backref, with the backref string and the backref properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known of the model

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Return the instance of the real field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

- **properties** – properties known of the model

Return type sqlalchemy relationship instance

init_expire_attributes (*registry, namespace, fieldname*)

Init dict of expiration properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

must_be_declared_as_attr ()

Return True, because it is a relationship

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

Exception FieldException

update_properties (*registry, namespace, fieldname, properties*)

Update the property use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.relationship.**Many2One** (***kwargs*)

Bases: anyblok.relationship.Relationship

Define a relationship attribute on the model

```
@register(Model)
class TheModel:

    relationship = Many2One(label="The relationship",
                             model=Model.RemoteModel,
                             remote_columns="The remote column",
                             column_names="The column which have the "
                                           "foreign key",
                             nullable=True,
                             unique=False,
                             one2many="themodels")
```

If the `remote_columns` are not define then, the system takes the primary key of the remote model

If the column doesn't exist, the column will be created. Use the nullable option. If the name is not filled, the name is “remote table'_remote column”

Parameters

- **model** – the remote model
- **remote_columns** – the column name on the remote model
- **column_names** – the column on the model which have the foreign key
- **nullable** – If the column_names is nullable
- **unique** – If True, add the unique constraint on the column
- **one2many** – create the one2many link with this many2one

apply_instrumentedlist (*registry, namespace, fieldname*)

Add the InstrumentedList class to replace List class as result of the query

Parameters **registry** – current registry

define_backref_properties (*registry, namespace, properties*)

Add in the backref_properties, new property for the backref

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **properties** – properties known of the model

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_backref (*registry, namespace, fieldname, properties*)

Create the real backref, with the backref string and the backref properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known of the model

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Parameters

- **registry** – current registry

- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Create the relationship

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **fieldname** – fieldname of the relationship
- **propertie** – the properties known

Return type Many2One relationship

init_expire_attributes (*registry, namespace, fieldname*)

Init dict of expiration properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

must_be_declared_as_attr ()

Return True, because it is a relationship

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

Exception FieldException

update_properties (*registry, namespace, fieldname, properties*)

Create the column which has the foreign key if the column doesn't exist

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **fieldname** – fieldname of the relationship
- **propertie** – the properties known

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.relationship.**One2One** (***kwargs*)
 Bases: anyblok.relationship.Many2One

Define a relationship attribute on the model

```
@register (Model)
class TheModel:

    relationship = One2One(label="The relationship",
                          model=Model.RemoteModel,
                          remote_columns="The remote column",
                          column_names="The column which have the "
                                      "foreign key",
                          nullable=False,
                          backref="themodels")
```

If the remote_columns are not define then, the system take the primary key of the remote model

If the column doesn't exist, then the column will be create. Use the nullable option. If the name is not filled then the name is "'remote table'_'remote column'"

Parameters

- **model** – the remote model
- **remote_columns** – the column name on the remote model
- **column_names** – the column on the model which have the foreign key
- **nullable** – If the column_names is nullable
- **backref** – create the one2one link with this one2one

apply_instrumentedlist (*registry, namespace, fieldname*)

Add the InstrumentedList class to replace List class as result of the query

Parameters **registry** – current registry

define_backref_properties (*registry, namespace, properties*)

Add option uselist = False

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **propertie** – the properties known

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_backref (*registry, namespace, fieldname, properties*)

Create the real backref, with the backref string and the backref properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

- **properties** – properties known of the model

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Create the relationship

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **fieldname** – fieldname of the relationship
- **property** – the properties known

Return type Many2One relationship

init_expire_attributes (*registry, namespace, fieldname*)

Init dict of expiration properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

must_be_declared_as_attr ()

Return True, because it is a relationship

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

Exception FieldException

update_properties (*registry, namespace, fieldname, properties*)

Create the column which has the foreign key if the column doesn't exist

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **fieldname** – fieldname of the relationship

- **propertie** – the properties known

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.relationship.**Many2Many** (***kwargs*)

Bases: anyblok.relationship.Relationship

Define a relationship attribute on the model

```
@register(Model)
class TheModel:

    relationship = Many2Many(label="The relationship",
                              model=Model.RemoteModel,
                              join_table="many2many table",
                              remote_columns="The remote column",
                              m2m_remote_columns="Name in many2many"
                              local_columns="local primary key"
                              m2m_local_columns="Name in many2many"
                              many2many="themodels")
```

if the **join_table** is not defined, then the table join is “join_’local table’_and_’remote table’”

Warning: The **join_table** must be filled when the declaration of the **Many2Many** is done in a **Mixin**

If the **remote_columns** are not define then, the system take the primary key of the remote model

if the **local_columns** are not define the take the **primary key of the local** model

Parameters

- **model** – the remote model
- **join_table** – the many2many table to join local and remote models
- **remote_columns** – the column name on the remote model
- **m2m_remote_columns** – the column name to remote model in m2m table
- **local_columns** – the column on the model
- **m2m_local_columns** – the column name to local model in m2m table
- **many2many** – create the opposite many2many on the remote model

apply_instrumentedlist (*registry, namespace, fieldname*)

Add the InstrumentedList class to replace List class as result of the query

Parameters **registry** – current registry

define_backref_properties (*registry, namespace, properties*)

Add in the backref_properties, new property for the backref

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **properties** – properties known of the model

forbid_instance (*cls*)

Raise an exception if the *cls* is an instance of this `__class__`

Parameters **cls** – instance of the class

Exception `FieldException`

format_backref (*registry, namespace, fieldname, properties*)

Create the real backref, with the backref string and the backref properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known of the model

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

Warning: In the case of the `get` is called in `classattribute`, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Create the relationship

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **fieldname** – fieldname of the relationship
- **properties** – the properties known

Return type `Many2One` relationship

init_expire_attributes (*registry, namespace, fieldname*)

Init dict of expiration properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

must_be_declared_as_attr()

Return True, because it is a relationship

must_be_duplicate_before_added()

Return False, it is the default value

native_type()

Return the native SQLAlchemy type

Exception FieldException

update_properties (*registry, namespace, fieldname, properties*)

Update the propertie use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

class anyblok.relationship.**One2Many** (***kwargs*)

Bases: anyblok.relationship.RelationShip

Define a relationship attribute on the model

```
@register(Model)
class TheModel:

    relationship = One2Many(label="The relationship",
                           model=Model.RemoteModel,
                           remote_columns="The remote column",
                           primaryjoin="Join condition"
                           many2one="themodel")
```

If the **primaryjoin** is not filled then the join condition is “‘local table’.local promary key’ == ‘remote table’.remote colum”

Parameters

- **model** – the remote model
- **remote_columns** – the column name on the remote model

- **primaryjoin** – the join condition between the remote column
- **many2one** – create the many2one link with this one2many

apply_instrumentedlist (*registry, namespace, fieldname*)

Add the InstrumentedList class to replace List class as result of the query

Parameters **registry** – current registry

define_backref_properties (*registry, namespace, properties*)

Add option in the backref if both model and remote model are the same, it is for the One2Many on the same model

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **property** – the properties known

find_foreign_key (*registry, properties, tablename*)

Return the primary key come from the first step property

Parameters

- **registry** – the registry which load the relationship
- **properties** – first step properties for the model
- **tablename** – the name of the table for the foreign key

Return type column name of the primary key

forbid_instance (*cls*)

Raise an exception if the cls is an instance of this __class__

Parameters **cls** – instance of the class

Exception FieldException

format_backref (*registry, namespace, fieldname, properties*)

Create the real backref, with the backref string and the backref properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known of the model

format_label (*fieldname*)

Return the label for this field

Parameters **fieldname** – if no label filled, the fieldname will be capitalized and returned

Return type the label for this field

get_property (*registry, namespace, fieldname, properties*)

Return the property of the field

<p>Warning: In the case of the get is called in classattribute, SQLAlchemy wrap for each call the column, the id of the wrapper is not the same</p>
--

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

get_sqlalchemy_mapping (*registry, namespace, fieldname, properties*)

Create the relationship

Parameters

- **registry** – the registry which load the relationship
- **namespace** – the name space of the model
- **fieldname** – fieldname of the relationship
- **property** – the properties known

Return type Many2One relationship

init_expire_attributes (*registry, namespace, fieldname*)

Init dict of expiration properties

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field

must_be_declared_as_attr ()

Return True, because it is a relationship

must_be_duplicate_before_added ()

Return False, it is the default value

native_type ()

Return the native SQLAlchemy type

Exception FieldException

update_properties (*registry, namespace, fieldname, properties*)

Update the property use to add new column

Parameters

- **registry** – current registry
- **namespace** – name of the model
- **fieldname** – name of the field
- **properties** – properties known to the model

wrap_expr_column (*fieldname*)

Return a default expr for the field

Parameters **fieldname** – name of the field

wrap_getter_column (*fieldname*)

Return a default getter for the field

Parameters **fieldname** – name of the field

anyblok._graphviz module

class anyblok._graphviz.**BaseSchema** (*name, format='png'*)
Common class extended by the type of schema

add_edge (*cls_1, cls_2, attr=None*)

Add new edge between 2 node

```
dot.add_edge(node1, node2)
```

Parameters

- **cls_1** – node (string or object) for the from
- **cls_2** – node (string or object) for the to

Paam attr attribute of the edge

render ()

Call graphviz to do the schema

save ()

render and create the output file

class anyblok._graphviz.**SQLSchema** (*name, format='png'*)
Create a schema to display the table model

```
dot = SQLSchema('the name of my schema')
t1 = dot.add_table('Table 1')
t1.add_column('c1', 'Integer')
t1.add_column('c2', 'Integer')
t2 = dot.add_table('Table 2')
t2.add_column('c1', 'Integer')
t2.add_foreign_key(t1, 'c2')
dot.save()
```

add_label (*name*)

Add a new node TableSchema without column

Parameters **name** – name of the table

Return type return the instance of TableSchema

add_table (*name*)

Add a new node TableSchema with column

Parameters **name** – name of the table

Return type return the instance of TableSchema

get_table (*name*)

Return the instance of TableSchema linked with the name of table

Parameters **name** – name of the table

Return type return the instance of TableSchema

class anyblok._graphviz.**TableSchema** (*name, parent, islabel=False*)
Describe one table

add_column (*name, type_, primary_key=False*)

Add a new column in the table

Parameters

- **name** – name of the column
- **type** – type of the column
- **primary_key** – if True, the string PK will be add

add_foreign_key (*node, label=None, nullable=True*)

Add a new foreign key

Parameters

- **node** – node (string or object) of the table linked
- **label** – name of the column of the foreign key
- **nullable** – bool to select the multiplicity of the association

render (*dot*)

Call graphviz to create the schema

class anyblok._graphviz.**ModelSchema** (*name, format='png'*)

Create a schema to display the UML model

```
dot = ModelSchema('The name of my UML schema')
cls = dot.add_class('My class')
cls.add_method('insert')
cls.add_property('items')
cls.add_column('my column')
dot.save()
```

add_class (*name*)

Add a new node ClassSchema with column

Parameters **name** – name of the class

Return type return the instance of ClassSchema

add_label (*name*)

Return the instance of ClassSchema linked with the name of class

Parameters **name** – name of the class

Return type return the instance of ClassSchema

get_class (*name*)

Add a new node ClassSchema without column

Parameters **name** – name of the class

Return type return the instance of ClassSchema

class anyblok._graphviz.**ClassSchema** (*name, parent, islabel=False*)

Use to display a class

add_column (*name*)

add a column in the class

Parameters **name** – name of the column

add_method (*name*)

add a method in the class

Parameters **name** – name of the method

add_property (*name*)

add a property in the class

Parameters **name** – name of the property

agregate (*node, label_from=None, multiplicity_from=None, label_to=None, multiplicity_to=None*)

add an edge with agregate shape to the node

Parameters

- **node** – node (string or object)
- **label_from** – attribute name
- **multiplicity_from** – multiplicity of the attribute
- **label_to** – attribute name
- **multiplicity_to** – multiplicity of the attribute

associate (*node, label_from=None, multiplicity_from=None, label_to=None, multiplicity_to=None*)

add an edge with associate shape to the node

Parameters

- **node** – node (string or object)
- **label_from** – attribute name
- **multiplicity_from** – multiplicity of the attribute
- **label_to** – attribute name
- **multiplicity_to** – multiplicity of the attribute

extend (*node*)

add an edge with extend shape to the node

Parameters **node** – node (string or object)

render (*dot*)

Call graphviz to do the schema

strong_agregate (*node, label_from=None, multiplicity_from=None, label_to=None, multiplicity_to=None*)

add an edge with strong agregate shape to the node

Parameters

- **node** – node (string or object)
- **label_from** – attribute name
- **multiplicity_from** – multiplicity of the attribute
- **label_to** – attribute name
- **multiplicity_to** – multiplicity of the attribute

anyblok.scripts module

anyblok.scripts.createdb (*application, configuration_groups, **kwargs*)

Create a database and install blok from config

Parameters

- **application** – name of the application
- **configuration_groups** – list configuration groupe to load
- ****kwargs** – ArgumentParser named arguments

`anyblok.scripts.updatedb(application, configuration_groups, **kwargs)`

Update an existing database

Parameters

- **application** – name of the application
- **configuration_groups** – list configuration groupe to load
- ****kwargs** – ArgumentParser named arguments

`anyblok.scripts.interpreter(application, configuration_groups, **kwargs)`

Execute a script or open an interpreter

Parameters

- **application** – name of the application
- **configuration_groups** – list configuration groupe to load
- ****kwargs** – ArgumentParser named arguments

`anyblok.scripts.run_exit(application, configuration_groups, **kwargs)`

Run nose unit test for the registry

Parameters

- **application** – name of the application
- **configuration_groups** – list configuration groupe to load
- ****kwargs** – ArgumentParser named arguments

`anyblok.scripts.cron_worker(application, configuration_groups, **kwargs)`

Execute a cron worker

Parameters

- **application** – name of the application
- **configuration_groups** – list configuration groupe to load
- ****kwargs** – ArgumentParser named arguments

`anyblok.scripts.registry2doc(application, configuration_groups, **kwargs)`

Return auto documentation for the registry

Parameters

- **application** – name of the application
- **configuration_groups** – list configuration groupe to load
- ****kwargs** – ArgumentParser named arguments

Contents

- *Helpers for unittest*
 - *TestCase*

- *DBTestCase*
- *BlokTestCase*
- *LogCapture*

Helpers for unittest

AnyBlok provides base test classes to help creating fixtures. Blok developers will be mostly interested in `BlokTestCase`. Base classes for unit/integration tests with anyblok.

This module provides `BlokTestCase`, which is the main one meant for blok tests, and `DBTestCase`, whose primary purpose is to test anyblok itself, in so-called “framework tests”.

TestCase

```
from anyblok.tests.testcase import TestCase
```

```
class anyblok.tests.testcase.TestCase(methodName='runTest')
```

```
    Bases: unittest.case.TestCase
```

Common helpers, not meant to be used directly.

```
    Configuration(**values)
```

Add Configuration value only in the contextmanager

```
    with TestCase.Configuration(db_name='a db name'):  
        self.assertEqual(Configuration.get('db_name'), 'a db name')
```

Parameters ****values** – values to update

```
    classmethod additional_setting()
```

```
    callCleanup()
```

```
    cleanup()
```

```
    classmethod createdb(keep_existing=False)
```

Create the database specified in configuration.

```
    cls.init_configuration_manager()  
    cls.createdb()
```

Parameters `keep_existing` – If false drop the previous db before create it

classmethod `dropdb()`

Drop the database specified in configuration.

```
cls.init_configuration_manager()  
cls.dropdb()
```

classmethod `getRegistry()`

Return the registry for the test database.

This assumes the database is created, and the registry has already been initialized:

```
registry = self.getRegistry()
```

Return type registry instance

classmethod `init_configuration_manager(**env)`

Initialise the configuration manager with environ variable to launch the test

Warning: For the moment we not use the environ variable juste constante

Parameters

- **prefix** – prefix the database name
- **env** – add another dict to merge with environ variable

setUp()

tearDown()

Roll back the session

DBTestCase

Warning: this testcase destroys the test database for each unittest

class `anyblok.tests.testcase.DBTestCase(methodName='runTest')`

Bases: `anyblok.tests.testcase.TestCase`

Base class for tests that need to work on an empty database.

Warning: The database is created and dropped with each unit test

For instance, this is the one used for Field, Column, Relationship, and more generally core framework tests.

The drawback of using this base class is that tests will be slow. The advantage is ultimate test isolation.

Sample usage:

```
from anyblok.tests.testcase import DBTestCase
```

```
def simple_column(ColumnType=None, **kwargs):

    @Declarations.register(Declarations.Model)
    class Test:

        id = Declarations.Column.Integer(primary_key=True)
        col = ColumnType(**kwargs)

class TestColumns(DBTestCase):

    def test_integer(self):
        Integer = Declarations.Column.Integer

        registry = self.init_registry(simple_column,
                                      ColumnType=Integer)
        test = registry.Test.insert(col=1)
        self.assertEqual(test.col, 1)
```

blok_entry_points = ('bloks',)
 setuptools entry points to load blok

init_registry (*function*, ***kwargs*)
 call a function to filled the blok manager with new model

Parameters

- **function** – function to call
- **kwargs** – kwargs for the function

Return type registry instance

setUp()
 Create a database and load the blok manager

classmethod setUpClass()
 Intialialise the configuration manager

tearDown()
 Clear the registry, unload the blok manager and drop the database

BlokTestCase

class anyblok.tests.testcase.**BlokTestCase** (*methodName='runTest'*)
 Bases: unittest.case.TestCase

Base class for tests meant to run on a preinstalled database.

The tests written with this class don't need to start afresh on a new database, and therefore run much faster than those inheriting DBTestCase. Instead, they expect the tested bloks to be already installed and up to date.

The session gets rollbacked after each test.

Such tests are appropriate for a typical blok developer workflow:

- create and install the bloks once
- run the tests of the blok under development repeatedly
- upgrade the bloks in database when needed (schema change, update of dependencies)

They are also appropriate for on the fly testing while installing the bloks: the tests of each blok get run on the database state they expect, before dependent (downstream) bloks, that could. e.g., alter the database schema, get themselves installed. This is useful to test a whole stack at once using only one database (typically in CI bots).

Sample usage:

```
from anyblok.tests.testcase import BlokTestCase

class MyBlokTest(BlokTestCase):

    def test_1(self):
        # access to the registry by ``self.registry``
        ...
```

classmethod additional_setting()

callCleanUp()

cleanUp()

registry = None

The instance of `anyblok.registry.Registry` to use in tests.

The `session_commit()` method is disabled to avoid side effects from one test to the other.

setUp()

classmethod setUpClass()

Initialize the registry.

tearDown()

Roll back the session

LogCapture

```
class anyblok.tests.testcase.LogCapture(names=None, install=True, level=1, propagate=None, attributes=('name', 'levelname', 'getMessage'), recursive_check=False)
```

Bases: `testfixtures.logcapture.LogCapture`

Overwrite `testfixtures.LogCapture` to add some helper methods

acquire()

Acquire the I/O thread lock.

actual()

addFilter(filter)

Add the specified filter to this handler.

atexit()

atexit_setup = False

check(*expected)

This will compare the captured entries with the expected entries provided and raise an `AssertionError` if they do not match.

Parameters expected – A sequence of 3-tuples containing the expected log entries. Each tuple should be of the form (logger_name, string_level, message)

clear()

Clear any entries that have been captured.

close()

Tidy up any resources used by the handler.

This version removes the handler from an internal map of handlers, `_handlers`, which is used for handler lookup by name. Subclasses should ensure that this gets called from overridden `close()` methods.

createLock()

Acquire a thread lock for serializing access to the underlying I/O.

emit(record)**filter(record)**

Determine if a record is loggable by consulting all the filters.

The default is to allow the record to be logged; any filter can veto this and the record is then dropped. Returns a zero value if a record is to be dropped, else non-zero.

Changed in version 3.2: Allow filters to be just callables.

flush()

Ensure all logging output has been flushed.

This version does nothing and is intended to be implemented by subclasses.

format(record)

Format the specified record.

If a formatter is set, use it. Otherwise, use the default formatter for the module.

get_critical_messages()

Return only the logging.CRITICAL messages

get_debug_messages()

Return only the logging.DEBUG messages

get_error_messages()

Return only the logging.ERROR messages

get_info_messages()

Return only the logging.INFO messages

get_messages(*levels)

Return the captured messages

```
with LogCapture() as logs:
    messages = logs.get_messages(INFO, WARNING)
```

Parameters **levels* – list of logging.level

Return type list of formatted message

get_name()**get_warning_messages()**

Return only the logging.WARNING messages

handle(record)

Conditionally emit the specified logging record.

Emission depends on filters which may have been added to the handler. Wrap the actual emission of the record with acquisition/release of the I/O thread lock. Returns whether the filter passed the record for emission.

handleError (*record*)

Handle errors which occur during an emit() call.

This method should be called from handlers when an exception is encountered during an emit() call. If raiseExceptions is false, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors. You could, however, replace this with a custom handler if you wish. The record which was being processed is passed in to this method.

install ()

Install this LogHandler into the Python logging framework for the named loggers.

This will remove any existing handlers for those loggers and drop their level to that specified on this LogCapture in order to capture all logging.

installed = False

instances = set()

name

release ()

Release the I/O thread lock.

removeFilter (*filter*)

Remove the specified filter from this handler.

setFormatter (*fmt*)

Set the formatter for this handler.

setLevel (*level*)

Set the logging level of this handler. level must be an int or a str.

set_name (*name*)

uninstall ()

Un-install this LogHandler from the Python logging framework for the named loggers.

This will re-instate any existing handlers for those loggers that were removed during installation and restore their level that prior to installation.

uninstall_all ()

This will uninstall all existing LogHandler objects.

Contents

- *Bloks*
 - *Blok anyblok-core*
 - * *Sequence*
 - * *Parameter*
 - * *API doc*
 - *Blok IO*
 - * *Mapping*

- * *Formatter*
- * *Exporter*
- * *Importer*
- * *API doc*
- *Blok IO CSV*
 - * *Exporter*
 - * *Importer*
 - * *API doc*
- *Blok IO XML*
 - * *Exporter*
 - * *Importer*
 - * *API doc*
- *Blok Model Auth*
 - * *API doc*

Blok anyblok-core

```
class anyblok.bloks.anyblok_core.AnyBlokCore(registry)
```

```
    Bases: anyblok.blok.Blok
```

This blok is required by all anyblok application. This blok define the main fonctionnality to install, update and uninstall blok. And also list the known models, fields, columns and relationships

```
    autoinstall = True
```

```
    conditional_by = []
```

```
    conflicting_by = []
```

```
    classmethod import_declaration_module()
```

```
    name = 'anyblok-core'
```

```
    optional_by = []
```

```
    pre_migration (latest_version)
```

```
    pre_migration_0_4_1_fields_become_polymorphic (latest_version)
```

```
    priority = 0
```

```
    classmethod reload_declaration_module (reload)
```

```
    required_by = ['anyblok-io']
```

```
    version = '0.9.6'
```

This blok is required by all anyblok application. This blok define the main fonctionnality to install, update and uninstall blok. And also list the known models, fields, columns and relationships:

- **Core Model**
 - Base: inherited by all the Model
 - SqlBase: Inherited only by the model with table

- SqlViewBase: Inherited only by the sql view model
- **System Models**
 - Blok: List the bloks
 - Model: List the models
 - Field: List of the fields
 - Column: List of the columns
 - Relationship: List of the relation ship
 - Sequence: Define database sequence
 - Parameter: Define application parameter

Sequence

Some behaviours need to have sequence:

```
sequence = registry.System.Sequence.insert(  
    code="string code",  
    formater="One prefix {seq} One suffix")
```

Note: It is a python formater, you can use the variable:

- seq: numero of the current data base sequence
 - code: code field
 - id: id field
-

Get the next value of the sequence:

```
sequence.nextval()
```

exemple:

```
seq = Sequence.insert(code='SO', formater="{code}-{seq:06d}")  
seq.nextval()  
>>> SO-000001  
seq.nextval()  
>>> SO-000002
```

Parameter

Parameter is a simple model key / value:

- key: must be a String
- value: any type

Add new value in the paramter model:

```
registry.System.Parameter.set(key, value)
```

Note: If the key already exist, then the value of the key is updated

Get an existing value:

```
registry.System.Parameter.get(key)
```

Warning: If the key does not existing then an ExceptionParameter will raise

Check the key exist:

```
registry.System.Parameter.is_exist(key)  # return a Boolean
```

Return and remove the parameter:

```
registry.System.Parameter.pop(key)
```

API doc

Authorization

class anyblok.bloks.anyblok_core.authorization.**Authorization**
 Bases: object

Namespace for models supporting authorization policies.

class anyblok.bloks.anyblok_core.authorization.**DefaultModelDeclaration**
 Bases: object

Pseudo model to represent the default value.

Core

class anyblok.bloks.anyblok_core.core.base.**Base**
 Bases: object

Inherited by all the models

Declaration type Core

Registry name Core.Base

classmethod **fire** (*event*, *args, **kwargs)
 Call a specific event on the model

Parameters **event** – Name of the event

classmethod **from_primary_keys** (**pks)
 No SQL Model has not primary key

classmethod **get_primary_keys** (**pks)
 No SQL Model has not primary key

classmethod **has_model_perm** (principals, permission)
 Check that one of principals has permission on given model.

Since this is a classmethod, even if called on a record, only its model class will be considered for the permission check.

has_perm (*principals, permission*)

Check that one of principals has permission on given record.

Since this is an ordinary instance method, it can't be used on the model class itself. For this use case, see

`has_model_perm()`

classmethod initialize_model ()

This method is called to initialize a model during the creation of the registry

to_primary_keys ()

No SQL Model has not primary key

class anyblok.bloks.anyblok_core.core.sqlbase.**SqlMixin**

Bases: object

classmethod aliased (**args, **kwargs*)

Facility to Apply an aliased on the model:

```
MyModelAliased = MyModel.aliased()
```

is equal at:

```
from sqlalchemy.orm import aliased
MyModelAliased = aliased(MyModel)
```

Return type SqlAlchemy aliased of the model

classmethod from_multi_primary_keys (**pks*)

return the instances of the model from the primary keys

Parameters **pks* – list of dict [{primary_key: value, ...}]

Return type instances of the model

classmethod from_primary_keys (***pks*)

return the instance of the model from the primary keys

Parameters ***pks* – dict {primary_key: value, ...}

Return type instance of the model

classmethod getFieldtype (*name*)

Return the type of the column

```
TheModel.getFieldtype(nameOfTheColumn)
```

this method take care if it is a polymorphic model or not

Parameters *name* – name of the column

Return type String, the name of the Type of column used

classmethod get_primary_keys ()

return the name of the primary keys of the model

Type list of the primary keys name

classmethod get_where_clause_from_primary_keys (***pks*)

return the where clause to find object from pks

Parameters ***pks* – dict {primary_key: value, ...}

Return type where clause

Exception SqlBaseException**classmethod** `query` (*elements)

Facility to do a SQLAlchemy query:

```
query = MyModel.query()
```

is equal at:

```
query = self.registry.session.query(MyModel)
```

Parameters **elements** – pass at the SQLAlchemy query, if the element is a string then that are see as field of the model

Return type SQLAlchemy Query

to_dict (*fields)

Transform a record to the dict of value

Parameters **fields** – list of fields to put in dict; if not selected, fields then take them all. A field is either one of these:

- a string (which is the name of the field)
- a 2-tuple if the field is a relationship (name of the field, tuple of foreign model fields)

Return type dict

Here are some examples:

```
=>> instance.to_dict() # get all fields
{"id": 1,
 "column1": "value 1",
 "column2": "value 2",
 "column3": "value 3",
 "relation1": {"relation_pk_1": 42, "relation_pk_2": "also 42"}
               # m2o or o2o : this is a dictionary
 "relation2": [{"id": 28}, {"id": 1}, {"id": 34}]
               # o2m or m2m : this is a list of dictionaries
}

=>> instance.to_dict("column1", "column2", "relation1")
               # get selected fields only (without any constraints)
{"column1": "value 1",
 "column2": "value 2",
 "relation1": {"relation_pk_1": 42, "relation_pk_2": "also 42"}
}

=>> instance.to_dict("column1", "column2", (
               # select fields to use in the relation related model
               "relation1", ("relation_pk1", "name", "value")
               # there is no constraints in the choice of fields
))
{"column1": "value",
 "column2": "value",
 "relation1": {"relation_pk_1": 42, "name": "H2G2", "value": "42"}
}

=>> instance.to_dict("column1", "column2", ("relation1", ))
# or
=>> instance.to_dict("column1", "column2", ("relation1", None))
```

```
# or
=>> instance.to_dict("column1", "column2", ("relation1", ()))
                                     # select all the fields of the relation ship
{"column1": "value",
 "column2": "value",
 "relation1": {"relation_pk_1": 42, "name": "H2G2", "value": "42"}
}

=>> instance.to_dict("column1", "column2", (
                                     # select relation fields recursively
    "relation1", ("name", "value", (
        "relation", ("a", "b", "c")
    ))
))
{"column1": "value",
 "column2": "value",
 "relation1": {"name": "H2G2", "value": "42", "relation": [
    {"a": 10, "b": 20, "c": 30},
    {"a": 11, "b": 22, "c": 33},
  ]}
}
```

to_primary_keys()

return the primary keys and values for this instance

Return type dict {primary key: value, ..}

class anyblok.bloks.anyblok_core.core.sqlbase.**SqlBase**

Bases: anyblok.bloks.anyblok_core.core.sqlbase.SqlMixin

this class is inherited by all the SQL model

Declaration type Core

Registry name Core.SqlBase

delete (*flush=True*)

Call the SQLAlchemy Query.delete method on the instance of the model:

```
self.delete()
```

is equal at:

```
flush the session
remove the instance of the session
and expire all the session, to reload the relation ship
```

expire (**fields*)

Expire the attribute of the instance, theses attributes will be load at the next call of the instance

see: http://docs.sqlalchemy.org/en/latest/orm/session_api.html #sqlalchemy.orm.session.Session.expire

expire_relationship_mapped (*mappers*)

Expire the objects linked with this object, in function of the mappers definition

find_relationship (**fields*)

Find column and relation ship link with the column or relationship passed in fields.

Parameters **fields* – lists of the attribute name

Return type list of the attribute name of the attribute and relation ship

Cached classmethod with size=128

classmethod insert (***kwargs*)

Insert in the table of the model:

```
MyModel.insert(...)
```

is equal at:

```
mymodel = MyModel(...)
MyModel.registry.session.add(mymodel)
MyModel.registry.flush()
```

classmethod multi_insert (**args*)

Insert in the table one or more entry of the model:

```
MyModel.multi_insert([ {...}, ...])
```

the flush will be done only one time at the end of the insert

Exception `SqlBaseException`

classmethod precommit_hook (*method, *args, **kwargs*)

Same in the registry a hook to call just before the commit

Warning: Only one instance of the hook is called before the commit

Parameters

- **method** – the method to call on this model
- **put_at_the_end_if_exist** – If `True` the hook is move at the end

refresh (**fields*)

Expire and reload all the attribute of the instance

See: http://docs.sqlalchemy.org/en/latest/orm/session_api.html #sqlalchemy.orm.session.Session.refresh

update (***values*)

Hight level method to update the session for the instance

```
self.update(val1=.., val2= ...)
```

..warning:

```
the columns and values is passed as named arguments to show
a difference with Query.update meth
```

class `anyblok.bloks.anyblok_core.core.sqlviewbase.SqlViewBase`

Bases: `anyblok.bloks.anyblok_core.core.sqlbase.SqlMixin`

this class is inherited by all the SQL view

Declaration type `Core`

Registry name `Core.SqlViewBase`

class `anyblok.bloks.anyblok_core.core.instrumentedlist.InstrumentedList`

Bases: `object`

class of the return of the `query.all()` or the relationship list

Declaration type Core

Registry name Core.InstrumentedList

class anyblok.bloks.anyblok_core.core.query.**Query** (*entities, session=None*)

Bases: sqlalchemy.orm.query.Query

Overload the SQLAlchemy Query

Declaration type Core

Registry name Core.Query

all ()

Return an instrumented list of the result of the query

with_perm (*principals, permission*)

Add authorization pre- and post-filtering to query.

This must be last in the construction chain of the query. Queries too complicated for the authorization system to infer safely will be refused.

Parameters

- **principals** – list, set or tuple of strings
- **permission** (*str*) – the permission to filter for

Returns a query-like object, with only the returning methods, such as `all()`, `count()` etc. available.

class anyblok.bloks.anyblok_core.core.session.**Session** (**args, **kwargs*)

Bases: sqlalchemy.orm.session.Session

Overload of the SQLAlchemy session

Declaration type Core

Registry name Core.Session

system

class anyblok.bloks.anyblok_core.system.**System**

Bases: object

Declaration type Model

Registry name Model.System

Tablename system

Inherit model or mixin

class anyblok.bloks.anyblok_core.system.blok.**Blok**

Bases: object

Declaration type Model

Registry name Model.System.Blok

Tablename system_blok

Inherit model or mixin

field name	Description
short_description	<ul style="list-style-type: none"> •Context: •Field type - <class 'any-blok.field.Function'> •Label - None •fget - get_short_description
state	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •selections: •toupdate - To update •installed - Installed •undefined - Undefined •toinstall - To install •touninstall - To uninstall •uninstalled - Uninstalled •Field type - <class 'any-blok.column.Selection'> •default - uninstalled •is crypted - False
installed_version	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64 •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
order	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.Integer'> •default - -1 •is crypted - False
name	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •primary_key - True •foreign_key - None •nullable - False •is crypted - False •Field type - <class 'any-

classmethod `apply_state (*bloks)`

Call the right method is the blok state change

Warning: for the uninstallation the method called is `uninstall_all`

Parameters `bloks` – list of the blok name load by the registry

classmethod `check_if_the_conditional_are_installed (blok)`

Return True if all the conditions to install the blok are satisfied

Parameters `blok` – blok name

Return type boolean

get_long_description ()

fget of the `long_description` Column.Selection

Return type the readme file of the blok

get_short_description ()

fget of the `short_description` Column.Selection

Return type the docstring of the blok

install ()

Method to install the blok

classmethod `list_by_state (*states)`

Return the blok name in function of the wanted states

Parameters `states` – list of the state

Return type list if state is a state, dict if the states is a list

load ()

Method to load the blok when the registry is completely loaded

classmethod `load_all ()`

Load all the installed bloks

uninstall ()

Method to uninstall the blok

classmethod `uninstall_all (*bloksname)`

Search and call the uninstall method for all the uninstalled bloks

Warning: Use the `desc` order to uninstall because we can't uninstall a dependencies before

Parameters `bloksname` – list of the blok name to uninstall

classmethod `update_list ()`

Populate the bloks list and update the state of existing bloks

upgrade ()

Method to update the blok

class `anyblok.bloks.anyblok_core.system.cache.Cache`

Bases: `object`

Declaration type `Model`

Registry name Model.System.Cache

Tablename system_cache

Inherit model or mixin

field name	Description
registry_name	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
id	<ul style="list-style-type: none"> •Context: •Label - None •autoincrement - True •primary_key - True •foreign_key - None •is crypted - False •Field type - <class 'any-blok.column.Integer'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
method	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False

classmethod clear_invalidate_cache()

Invalidate the cache that needs to be invalidated

classmethod detect_invalidation()

Return True if a new invalidation is found in the table

Return type Boolean

classmethod get_invalidation()

Return the pointer of the method to invalidate

classmethod `get_last_id()`

Return the last primary key `id` value

classmethod `initialize_model()`

Initialize the `last_cache_id` known

classmethod `invalidate(registry_name, method)`

Call the invalidation for a specific method cached on a model

Parameters

- **registry_name** – namespace of the model
- **method** – name of the method on the model

Exception `CacheException`

class `anyblok.bloks.anyblok_core.system.field.Field`

Bases: `object`

Declaration type `Model`

Registry name `Model.System.Field`

Tablename `system_field`

Inherit model or mixin

field name	Description
ftype	<ul style="list-style-type: none"> •Context: •size - 64 •Label - Type •DB column name - None •foreign_key - None •nullable - True •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
model	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
code	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - True •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
name	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
entity_type	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - True
8.1. Blok anyblok-core	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - True

classmethod `add_field` (*rname, label, model, table, ftype*)

Insert a field definition

Parameters

- **rname** – name of the field
- **label** – label of the field
- **model** – namespace of the model
- **table** – name of the table of the model
- **ftype** – type of the AnyBlok Field

classmethod `alter_field` (*field, label, ftype*)

Update an existing field

Parameters

- **field** – instance of the Field model to update
- **label** – label of the field
- **ftype** – type of the AnyBlok Field

class `anyblok.bloks.anyblok_core.system.column.Column`

Bases: `anyblok.model.Field`

Declaration type Model

Registry name Model.System.Column

Tablename system_column

Inherit model or mixin

- <class 'anyblok.model.Field'>

field name	Description
remote_model	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64 •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
name	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
model	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
primary_key	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.Boolean'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
foreign_key	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64
8.1. Blok anyblok-core	<ul style="list-style-type: none"> •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'>

classmethod `add_field` (*cname, column, model, table, ftype*)

Insert a column definition

Parameters

- **cname** – name of the column
- **column** – instance of the column
- **model** – namespace of the model
- **table** – name of the table of the model
- **ftype** – type of the AnyBlok Field

classmethod `alter_field` (*column, meta_column, ftype*)

Update an existing column

Parameters

- **column** – instance of the Column model to update
- **meta_column** – instance of the SQLAlchemy column
- **ftype** – type of the AnyBlok Field

classmethod `get_cname` (*field, cname*)

Return the real name of the column

Parameters

- **field** – the instance of the column
- **cname** – Not use here

Return type string of the real column name

class `anyblok.bloks.anyblok_core.system.relationship.Relationship`

Bases: `anyblok.model.Field`

Declaration type Model

Registry name Model.System.Relationship

Tablename system_relationship

Inherit model or mixin

- <class 'anyblok.model.Field'>

field name	Description
remote_model	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
remote_column	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64 •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
model	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
nullable	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.Boolean'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
local_column	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64
8.1. Blok anyblok-core	<ul style="list-style-type: none"> •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'>

classmethod `add_field(rname, relation, model, table, ftype)`

Insert a relationship definition

Parameters

- **rname** – name of the relationship
- **relation** – instance of the relationship
- **model** – namespace of the model
- **table** – name of the table of the model
- **ftype** – type of the AnyBlok Field

class `anyblok.bloks.anyblok_core.system.cron.Cron`

Bases: `object`

Declaration type `Model`

Registry name `Model.System.Cron`

Tablename `system_cron`

Inherit model or mixin

class `anyblok.bloks.anyblok_core.system.cron.Job`

Bases: `object`

Declaration type `Model`

Registry name `Model.System.Cron.Job`

Tablename `system_cron_job`

Inherit model or mixin

field name	Description
available_at	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.DateTime'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
done_at	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.DateTime'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
model	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - Model.System.Model => name •nullable - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
is_a_class_method	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.Boolean'> •default - True •is crypted - False
error	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.Text'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
8.1. Blok anyblok-core	157
params	<ul style="list-style-type: none"> •Context:

class anyblok.bloks.anyblok_core.system.cron.**Worker** (*job*)

Bases: threading.Thread

Declaration type Model

Registry name Model.System.Cron.Worker

Tablename system_cron_worker

Inherit model or mixin

class anyblok.bloks.anyblok_core.system.model.**Model**

Bases: object

Models assembled

Declaration type Model

Registry name Model.System.Model

Tablename system_model

Inherit model or mixin

field name	Description
is_sql_model	<ul style="list-style-type: none"> •Context: •Label - Is a SQL model •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.Boolean'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
description	<ul style="list-style-type: none"> •Context: •Field type - <class 'any-blok.field.Function'> •Label - None •fget - get_model_doc_string
name	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 256 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
table	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 256 •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False

get_model_doc_string()

Return the docstring of the model

classmethod update_list()

Insert and update the table of models

Exception Exception

class anyblok.bloks.anyblok_core.system.parameter.**Parameter**

Bases: object

System Parameter

Declaration type Model**Registry name** Model.System.Parameter**Tablename** system_parameter**Inherit model or mixin**

field name	Description
key	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
multi	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class 'any-blok.column.Boolean'> •default - False •is crypted - False
value	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.Json'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False

classmethod `get (key)`

Return the value of the key

Parameters **key** – key to check**Return type** return value**Exception** ExceptionParameter**classmethod** `is_exist (key)`

Check if one parameter exist for the key

Parameters **key** – key to check

Return type Boolean, True if exist

classmethod **pop** (*key*)

Remove return the value of the key

Parameters **key** – key to check

Return type return value

Exception ExceptionParameter

classmethod **set** (*key, value*)

Insert or Update parameter for the key

Parameters

- **key** – key to save
- **value** – value to save

class `anyblok.bloks.anyblok_core.system.sequence.Sequence`

Bases: `object`

System sequence

Declaration type Model

Registry name Model.System.Sequence

Tablename system_sequence

Inherit model or mixin

field name	Description
number	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.Integer'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
code	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
seq_name	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False
id	<ul style="list-style-type: none"> •Context: •Label - None •autoincrement - True •primary_key - True •foreign_key - None •is crypted - False •Field type - <class 'any-blok.column.Integer'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
formater	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None
162	<ul style="list-style-type: none"> •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'any-


```

classmethod create_sequence (values)
    create the sequence for one instance

classmethod initialize_model ()
    Create the sequence to determine name

classmethod insert (**kwargs)
    Overwrite insert

classmethod multi_insert (*args)
    Overwrite multi_insert

nextval ()
    return the next value of the sequence

classmethod nextvalBy (**kwargs)
    Get the first sequence filtering by entries and return the next value

```

documentation

```

class anyblok.bloks.anyblok_core.documentation.DocElement
    Bases: object

```

Declaration type Mixin

Inherit model or mixin

```

class anyblok.bloks.anyblok_core.documentation.Documentation
    Bases: anyblok.mixin.DocElement

```

Declaration type Model

Registry name Model.Documentation

Tablename documentation

Inherit model or mixin

- <class 'anyblok.mixin.DocElement'>

```

class anyblok.bloks.anyblok_core.documentation.blok.Blok (blok)
    Bases: object

```

Declaration type Model

Registry name Model.Documentation.Blok

Tablename documentation_blok

Inherit model or mixin

```

class anyblok.bloks.anyblok_core.documentation.model.Model (model)
    Bases: anyblok.mixin.DocElement

```

Declaration type Model

Registry name Model.Documentation.Model

Tablename documentation_model

Inherit model or mixin

- <class 'anyblok.mixin.DocElement'>

```

class anyblok.bloks.anyblok_core.documentation.model.attribute.Attribute (attribute)
    Bases: object

```

Declaration type Model

Registry name Model.Documentation.Model.Attribute

Tablename documentation_model_attribute

Inherit model or mixin

class anyblok.bloks.anyblok_core.documentation.model.field.**Field** (*field*)

Bases: object

Declaration type Model

Registry name Model.Documentation.Model.Field

Tablename documentation_model_field

Inherit model or mixin

exception

exception anyblok.bloks.anyblok_core.exceptions.**CoreBaseException**

Bases: TypeError

Exception for Core.Base

exception anyblok.bloks.anyblok_core.exceptions.**SqlBaseException**

Bases: Exception

Simple Exception for sql base

exception anyblok.bloks.anyblok_core.exceptions.**QueryException**

Bases: Exception

Simple Exception for query

exception anyblok.bloks.anyblok_core.exceptions.**CacheException**

Bases: Exception

Simple Exception for the cache Model

exception anyblok.bloks.anyblok_core.exceptions.**ParameterException**

Bases: Exception

Simple exception for System.Parameter

exception anyblok.bloks.anyblok_core.exceptions.**CronWorkerException**

Bases: Exception

Simple exception for System.Parameter

Blok IO

class anyblok.bloks.io.**AnyBlokIO** (*registry*)

Bases: anyblok.blok.Blok

In / Out tool's:

- Formater: convert value 2 str or str 2 value in function of the field,
- Importer: main model to define an import,
- Exporter: main model to define an export,

conditional_by = []

conflicting_by = []

```

classmethod declare_io()
classmethod import_declaration_module()
name = 'anyblok-io'
optional_by = []
classmethod reload_declaration_module(reload)
required = ['anyblok-core']
required_by = ['anyblok-io-csv', 'anyblok-io-xml']
version = '0.9.6'

```

Note: Require the anyblok-io blok

Mapping

Model.IO.Mapping allows to link a Model instance by a Model namespace and str key. this key is an external *ID*

Save an instance with a key:

```

Blok = self.registry.System.Blok
blok = Blok.query().filter(Blok.name == 'anyblok-core').first()
self.registry.IO.Mapping.set('External ID', blok)

```

Warning: By default if you save another instance with the same key and the same model, an `IOMappingSetException` will be raised. If really you want this mapping you must call the set method with the named argument `raiseifexist=False`:

```

self.registry.IO.Mapping.set('External ID', blok, raiseifexist=False)

```

Get an entry in the mapping:

```

blok2 = self.registry.IO.Mapping.get('Model.System.Blok', 'External ID')
assert blok2 is blok

```

Formater

The goal of the formater is to get:

- value from string
- value from mapping key
- string from value
- mapping key from value

The value is the value of the field.

Warning: The relationships are particular cases. The value is the json of the primary keys. The Many2Many and the One2Many are the json of the list of the primary keys

Exporter

The `Model.IO.Exporter` export some entries in function of configuration. `anyblok-io` blok doesn't give complete exporter, just the base Model to standardize all the possible export:

```
exporter = registry.IO.Exporter.insert(...) # create a exporter
entries = ... # entries are instance of model
fp = exporter.run(entries)
# fp is un handler on the opened file (StringIO)
```

Importer

The `Model.IO.Importer` import some entries in function of configuration. `anyblok-io` blok doesn't give complete importer, just the base Model to standardize all the possible import:

```
importer = registry.IO.Importer.insert(...) # create an importer
# the file to import are filled in the parameter
entries = importer.run()
```

API doc

exceptions

exception `anyblok.bloks.io.exceptions.IOException`

Bases: `Exception`

IO exception

exception `anyblok.bloks.io.exceptions.IOMappingCheckException`

Bases: `anyblok.bloks.io.exceptions.IOException`

IO Exception for setter

exception `anyblok.bloks.io.exceptions.IOMappingSetException`

Bases: `anyblok.bloks.io.exceptions.IOException`

IO Exception for setter

exception `anyblok.bloks.io.exceptions.ImporterException`

Bases: `Exception`

Simple Exception for importer

exception `anyblok.bloks.io.exceptions.ExporterException`

Bases: `Exception`

Simple Exception for exporter

exception `anyblok.bloks.io.exceptions.FormatterException`

Bases: `Exception`

Simple Exception for importer

mapping**class** `anyblok.bloks.io.mapping.Mapping`Bases: `object`**Declaration type** `Model`**Registry name** `Model.IO.Mapping`**Tablename** `io_mapping`**Inherit model or mixin**

field name	Description
key	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'anyblok.column.String'> •default - <class 'anyblok.column.NoDefaultValue'> •DB column name - None
model	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - Model.System.Model => name •size - 64 •is crypted - False •Field type - <class 'anyblok.column.String'> •default - <class 'anyblok.column.NoDefaultValue'> •DB column name - None
primary_key	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'anyblok.column.Json'> •default - <class 'anyblok.column.NoDefaultValue'> •is crypted - False

classmethod `check_primary_keys(model, *pks)`

check if the all the primary keys match with primary keys of the model

Parameters

- **model** – model to check
- **pks** – list of the primary keys to check

Exception `IOMappingCheckException`

classmethod `delete` (*model, key, mapping_only=True*)

Delete the key for this model

Parameters

- **model** – model of the mapping
- **key** – string of the key

Return type Boolean True if the mapping is removed

filter_by_model_and_key (*model, key*)

SQLAlchemy hybrid method to filter by model and key

Parameters

- **model** – model of the mapping
- **key** – external key of the mapping

Hybrid method

filter_by_model_and_keys (*model, *keys*)

SQLAlchemy hybrid method to filter by model and key

Parameters

- **model** – model of the mapping
- **key** – external key of the mapping

Hybrid method

classmethod `get` (*model, key*)

return instance of the model with this external key

Parameters

- **model** – model of the mapping
- **key** – string of the key

Return type instance of the model

classmethod `get_mapping_primary_keys` (*model, key*)

return primary key for a model and an external key

Parameters

- **model** – model of the mapping
- **key** – string of the key

Return type dict primary key: value or None

classmethod `multi_delete` (*model, *keys, **kwargs*)

Delete all the keys for this model

Parameters

- **model** – model of the mapping
- ***keys** – list of the key

Return type Boolean True if the mappings are removed

classmethod `set` (*key, instance, raiseifexist=True*)

Add or update a mmping with a model and a external key

Parameters

- **model** – model to check
- **key** – string of the key
- **instance** – instance of the model to save
- **raiseifexist** – boolean (True by default), if True and the entry exist then an exception is raised

Exception IOMappingSetException

classmethod `set_primary_keys` (*model, key, pks, raiseifexist=True*)

Add or update a mmping with a model and a external key

Parameters

- **model** – model to check
- **key** – string of the key
- **pks** – dict of the primary key to save
- **raiseifexist** – boolean (True by default), if True and the entry exist then an exception is raised

Exception IOMappingSetException

mixin

class `anyblok.bloks.io.mixin.IOMixin`

Bases: `object`

Declaration type Mixin

Inherit model or mixin

field name	Description
mode	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •selections - get_mode_choices •Field type - <class 'anyblok.column.Selection'> •default - <class 'anyblok.column.NoDefaultValue'> •is crypted - False
model	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - Model.System.Model => name •nullable - False •Field type - <class 'anyblok.column.String'> •default - <class 'anyblok.column.NoDefaultValue'> •is crypted - False
id	<ul style="list-style-type: none"> •Context: •Label - None •autoincrement - True •primary_key - True •foreign_key - None •is crypted - False •Field type - <class 'anyblok.column.Integer'> •default - <class 'anyblok.column.NoDefaultValue'> •DB column name - None

importer

class anyblok.bloks.io.importer.**Importer**

Bases: anyblok.mixin.IOMixin

Declaration type Model

Registry name Model.IO.Importer

Tablename io_importer

Inherit model or mixin

- <class 'anyblok.mixin.IOMixin'>

field name	Description
offset	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class <p>‘anyblok.column.Integer’></p> <ul style="list-style-type: none"> • default - 0 • is crypted - False
check_import	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class <p>‘anyblok.column.Boolean’></p> <ul style="list-style-type: none"> • default - False • is crypted - False
nb_grouped_lines	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class <p>‘anyblok.column.Integer’></p> <ul style="list-style-type: none"> • default - 50 • is crypted - False
commit_at_each_grouped	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •Field type - <class <p>‘anyblok.column.Boolean’></p> <ul style="list-style-type: none"> • default - True • is crypted - False
file_to_import	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class <p>‘anyblok.column.LargeBinary’></p> <ul style="list-style-type: none"> • default - <class <p>‘anyblok.column.NoDefaultValue’></p> <ul style="list-style-type: none"> • is crypted - False

exporter

class anyblok.bloks.io.exporter.**Exporter**
Bases: anyblok.mixin.IOMixin

Declaration type Model

Registry name Model.IO.Exporter

Tablename io_exporter

Inherit model or mixin

- <class 'anyblok.mixin.IOMixin'>

formater

class anyblok.bloks.io.formater.**Formater**
Bases: object

Declaration type Model

Registry name Model.IO.Formater

Tablename io_formater

Inherit model or mixin

class anyblok.bloks.io.formater.**Float**
Bases: anyblok.model.Formater

Declaration type Model

Registry name Model.IO.Formater.Float

Tablename io_formater_float

Inherit model or mixin

- <class 'anyblok.model.Formater'>

class anyblok.bloks.io.formater.**Decimal**
Bases: anyblok.model.Formater

Declaration type Model

Registry name Model.IO.Formater.Decimal

Tablename io_formater_decimal

Inherit model or mixin

- <class 'anyblok.model.Formater'>

class anyblok.bloks.io.formater.**Json**
Bases: anyblok.model.Formater

Declaration type Model

Registry name Model.IO.Formater.Json

Tablename io_formater_json

Inherit model or mixin

- <class 'anyblok.model.Formater'>

class anyblok.bloks.io.formater.**Interval**
Bases: anyblok.model.Formater

Declaration type Model

Registry name Model.IO.Formatter.Interval

Tablename io_formatter_interval

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

class anyblok.bloks.io.formatter.**Integer**

Bases: anyblok.model.Formatter

Declaration type Model

Registry name Model.IO.Formatter.Integer

Tablename io_formatter_integer

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

class anyblok.bloks.io.formatter.**SmallInteger**

Bases: anyblok.model.Integer

Declaration type Model

Registry name Model.IO.Formatter.SmallInteger

Tablename io_formatter_smallinteger

Inherit model or mixin

- <class 'anyblok.model.Integer'>

class anyblok.bloks.io.formatter.**BigInteger**

Bases: anyblok.model.Integer

Declaration type Model

Registry name Model.IO.Formatter.BigInteger

Tablename io_formatter_biginteger

Inherit model or mixin

- <class 'anyblok.model.Integer'>

class anyblok.bloks.io.formatter.**Boolean**

Bases: anyblok.model.Formatter

Declaration type Model

Registry name Model.IO.Formatter.Boolean

Tablename io_formatter_boolean

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

class anyblok.bloks.io.formatter.**Time**

Bases: anyblok.model.Formatter

Declaration type Model

Registry name Model.IO.Formatter.Time

Tablename io_formatter_time

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

class anyblok.bloks.io.formater.**Date**

Bases: anyblok.model.Formatter

Declaration type Model

Registry name Model.IO.Formatter.Date

Tablename io_formatter_date

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

class anyblok.bloks.io.formater.**DateTime**

Bases: anyblok.model.Formatter

Declaration type Model

Registry name Model.IO.Formatter.DateTime

Tablename io_formatter_datetime

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

class anyblok.bloks.io.formater.**Many2One**

Bases: anyblok.model.Formatter

Declaration type Model

Registry name Model.IO.Formatter.Many2One

Tablename io_formatter_many2one

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

class anyblok.bloks.io.formater.**One2One**

Bases: anyblok.model.Many2One

Declaration type Model

Registry name Model.IO.Formatter.One2One

Tablename io_formatter_one2one

Inherit model or mixin

- <class 'anyblok.model.Many2One'>

class anyblok.bloks.io.formater.**Many2Many**

Bases: anyblok.model.Formatter

Declaration type Model

Registry name Model.IO.Formatter.Many2Many

Tablename io_formatter_many2many

Inherit model or mixin

- <class 'anyblok.model.Formatter'>

```
class anyblok.bloks.io.formater.One2Many
    Bases: anyblok.model.Many2Many

    Declaration type Model

    Registry name Model.IO.Formatter.One2Many

    Tablename io_formatter_one2many

    Inherit model or mixin
        • <class 'anyblok.model.Many2Many'>
```

Blok IO CSV

```
class anyblok.bloks.io_csv.AnyBlokIOCSV(registry)
    Bases: anyblok.blok.Blok

    CSV Importer / Exporter behaviour

    conditional_by = []

    conflicting_by = []

    classmethod import_declaration_module()

    name = 'anyblok-io-csv'

    optional_by = []

    classmethod reload_declaration_module(reload)

    required = ['anyblok-io']

    required_by = []

    version = '0.9.6'
```

Note: Require the anyblok-io-csv blok

Exporter

Add an exporter mode (CSV) in AnyBlok:

```
Exporter = registry.IO.Exporter.CSV
```

Define what export:

```
csv_delimiter = ','
csv_quotchar = '"'
model = ``Existing model name``
fields = [
    {'name': 'field name'},
    {'name': 'fieldname1.fieldname2. ... .fieldnamen'} # fieldname1, fieldname 2_
    ↳ must be Many2One or have foreign keys
    {'name': 'field', model="external_id"} # field must be Many2One or foreign_keys_
    ↳ or primary keys
    ...
]
```

Create the Exporter:

```
exporter = Exporter.insert(csv_delimiter=csv_delimiter,
                           csv_quotechar=csv_quotechar,
                           model=model,
                           fields=fields)
```

Warning: You can also make insert with `registry.IO.Exporter` directly

Run the export:

```
fp = exporter.run(entries) # entries are instance of the ``model``
```

Importer

Add an importer mode (CSV) in AnyBlok:

```
Importer = registry.IO.Importer.CSV
```

Define what import:

```
csv_delimiter = ','
csv_quotechar = '"'
model = ``Existing model name``
with open(..., 'rb') as fp:
    file_to_import = fp.read()
```

Create the Importer:

```
importer = Importer.insert(csv_delimiter=csv_delimiter,
                            csv_quotechar=csv_quotechar,
                            model=model,
                            file_to_import=file_to_import)
```

Warning: You can also make insert with `registry.IO.Importer` directly

Run the import:

```
res = importer.run()
```

The result is a dict with:

- `error_found`: List the error, durring the import
- `created_entries`: Entries created by the import
- `updated_entries`: Entries updated by the import

List of the options for the import:

- **csv_on_error:**
 - `raise_now`: Raise now

- `raise_at_the_end` (default): Raise at the end
 - `ignore`: Ignore and continue
- **`csv_if_exist`:**
 - `pass`: Pass to the next record
 - `overwrite` (default): Update the record
 - `create`: Create another record
 - `raise`: Raise an exception
- **`csv_if_does_not_exist`:**
 - `pass`: Pass to the next record
 - `create` (default): Create another record
 - `raise`: Raise an exception

API doc

exceptions

exception `anyblok.bloks.io_csv.exceptions.CSVImporterException`

Bases: `anyblok.bloks.io.exceptions.ImporterException`

Simple exception for CSV importer

exception `anyblok.bloks.io_csv.exceptions.CSVExporterException`

Bases: `anyblok.bloks.io.exceptions.ExporterException`

Simple exception for CSV exporter

mixin

class `anyblok.bloks.io_csv.mixin.IOCSVFieldMixin`

Bases: `object`

Declaration type Mixin

Inherit model or mixin

field name	Description
name	<ul style="list-style-type: none">•Context:•size - 64•Label - None•DB column name - None•foreign_key - None•nullable - False•Field type - <class 'any-blok.column.String'>•default - <class 'any-blok.column.NoDefaultValue'>•is crypted - False
id	<ul style="list-style-type: none">•Context:•Label - None•autoincrement - True•primary_key - True•foreign_key - None•is crypted - False•Field type - <class 'any-blok.column.Integer'>•default - <class 'any-blok.column.NoDefaultValue'>•DB column name - None

class anyblok.bloks.io_csv.mixin.**IOCSVMixin**

Bases: object

Declaration type Mixin

Inherit model or mixin

field name	Description
csv_delimiter	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'anyblok.column.String'> •default - , •is crypted - False
csv_quotechar	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •Field type - <class 'anyblok.column.String'> •default - “ •is crypted - False

importer

class anyblok.bloks.io_csv.importer.**Importer**

Bases: anyblok.mixin.IOCSVMixin

Declaration type Model

Registry name Model.IO.Importer

Tablename io_importer

Inherit model or mixin

- <class 'anyblok.mixin.IOCSVMixin'>

field name	Description
csv_if_does_not_exist	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64 •selections: •('pass', 'Pass to the next record') •('create', 'Create the record') •('raise', 'Raise an exception') •Field type - <class <p>'anyblok.column.Selection'></p> <ul style="list-style-type: none"> • default - create • is crypted - False
csv_on_error	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64 •selections: •('raise_now', 'Raise now') •('raise_at_the_end', 'Raise at the end') •('ignore', 'Ignore and continue') •Field type - <class <p>'anyblok.column.Selection'></p> <ul style="list-style-type: none"> • default - raise_at_the_end • is crypted - False
csv_if_exist	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64 •selections: •('pass', 'Pass to the next record') •('overwrite', 'Update the record') •('create', 'Create another record') •('raise', 'Raise an exception') •Field type - <class <p>'anyblok.column.Selection'></p> <ul style="list-style-type: none"> • default - overwrite • is crypted - False

class anyblok.bloks.io_csv.importer.**CSV** (*importer*)

Bases: object

Declaration type Model

Registry name Model.IO.Importer.CSV

Tablename io_importer_csv

Inherit model or mixin

exporter

class anyblok.bloks.io_csv.exporter.**Exporter**

Bases: anyblok.mixin.IOCSVMixin

Declaration type Model

Registry name Model.IO.Exporter

Tablename io_exporter

Inherit model or mixin

- <class 'anyblok.mixin.IOCSVMixin'>

class anyblok.bloks.io_csv.exporter.**Field**

Bases: anyblok.mixin.IOCSVFieldMixin

Declaration type Model

Registry name Model.IO.Exporter.Field

Tablename io_exporter_field

Inherit model or mixin

- <class 'anyblok.mixin.IOCSVFieldMixin'>

field name	Description
mode	<ul style="list-style-type: none"> •Context: •size - 64 •Label - None •DB column name - None •foreign_key - None •nullable - False •selections - get_selection •Field type - <class 'any-blok.column.Selection'> •default - any •is crypted - False
exporter	<ul style="list-style-type: none"> •Context: •Label - None •backref - fields_to_export •model - Model.IO.Exporter •Field type - <class 'any-blok.relationship.Many2One'> •unique - False •info: •remote_model - Model.IO.Exporter •remote_name - fields_to_export •nullable - False •_column_names - None •_remote_columns - None
mapping	<ul style="list-style-type: none"> •Context: •Label - None •DB column name - None •foreign_key - None •size - 64 •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •is crypted - False

class anyblok.bloks.io_csv.exporter.CSV (*exporter*)

Bases: object

Declaration type Model

Registry name Model.IO.Exporter.CSV

Tablename io_exporter_csv

Inherit model or mixin

Blok IO XML

class anyblok.bloks.io_xml.**AnyBlokIOXML**(registry)

Bases: anyblok.blok.Blok

XML Importer / Exporter behaviour

Warning: Importer and Exporter are not implemented yet

```
conditional_by = []
conflicting_by = []
classmethod import_declaration_module()
name = 'anyblok-io-xml'
optional_by = []
classmethod reload_declaration_module(reload)
required = ['anyblok-io']
required_by = []
version = '0.9.6'
```

Note: Require the anyblok-io-xml blok

Exporter

TODO

Importer

Add an importer mode (XML) in AnyBlok:

```
Importer = registry.IO.Importer.XML
```

Define what import:

```
model = ``Existing model name``
with open(..., 'rb') as fp:
    file_to_import = fp.read()
```

Create the Importer:

```
importer = Importer.insert(model=model,
                           file_to_import=file_to_import)
```

Warning: You can also make insert with registry.IO.Importer directly

Run the import:

```
res = importer.run()
```

The result is a dict with:

- `error_found`: List the error, durring the import
- `created_entries`: Entries created by the import
- `updated_entries`: Entries updated by the import

Root structure of the XML file:

```
<records on_error="...">
    ...
</records>
```

`raise` can have the value:

- `raise` (dafault)
- `ignore`

`records` node can have:

- `commit`: commit the import, only if no error found
- `record`: import one record

Add a record:

```
<records>
  <record>
    ...
    <field name="..." />
    ...
  </record>
</records>
```

Record attribute:

- `model`: if not filled, then the importer will indicate the model
- `external_id`: Mapping key
- `param`: Mapping key only for the import (not save)
- **`on_error`:**
 - `raise`
 - `ignore` (default)
- **`if_exist`:**
 - `overwrite` (default)
 - `create`
 - `pass`: continue to the next record
 - `continue`: continue on the sub record without take this record
 - `raise`
- **`id_does_not_exist`:**
 - `create` (default)

- pass
- raise

The field node represents a Field, a Column or Relationship, the attributes are:

- **name** (required): name of the field, column or relationship

Case of the relationship, they have some more attributes:

- **external_id:**
- **param:**
- **on_error:**
 - raise
 - ignore (default)
- **if_exist:**
 - overwrite (default)
 - create
 - pass: continue to the next record
 - continue: continue on the sub record without take this record
 - raise
- **id_does_not_exist:**
 - create (default)
 - pass
 - raise

Many2One and One2One declaration is directly in the field node:

```
<records
  <record
    ...
    <field name="Many2One or One2One">
      ...
      <field name="..." />
      ...
    </field>
    ...
  </record>
</records>
```

One2Many and Many2Many declarations are also in the field but with a record node:

```
<records
  <record
    ...
    <field name="Many2Many or One2Many">
      ...
      <record>
        ...
        <field name="..." />
        ...
      </record>
    </field>
  </record>
</records>
```

```
        ...
    </field>
    ...
</record>
</records>
```

In the case of polymorphisme you may use the attribute model on record:

```
<records
  <record
    ...
    <field name="Many2Many or One2Many">
      ...
      <record model="`polymorphic model`">
        ...
        <field name="..." />
        ...
      </record>
      <record model="`another polymorphic model`">
        ...
        <field name="..." />
        ...
      </record>
    ...
  </field>
  ...
</record>
</records>
```

API doc

exceptions

exception `anyblok.bloks.io_xml.exceptions.XMLImporterException`

Bases: `anyblok.bloks.io.exceptions.ImporterException`

Simple exception for XML importer

exception `anyblok.bloks.io_xml.exceptions.XMLElementException`

Bases: `anyblok.bloks.io.exceptions.ExporterException`

Simple exception for XML exporter

importer

class `anyblok.bloks.io_xml.importer.Importer`

Bases: `object`

Declaration type `Model`

Registry name `Model.IO.Importer`

Tablename `io_importer`

Inherit model or mixin

class `anyblok.bloks.io_xml.importer.XML(importer)`

Bases: `object`

Declaration type `Model`

Registry name Model.IO.Importer.XML

Tablename io_importer_xml

Inherit model or mixin

exporter

class anyblok.bloks.io_xml.exporter.**Exporter**
Bases: object

Declaration type Model

Registry name Model.IO.Exporter

Tablename io_exporter

Inherit model or mixin

class anyblok.bloks.io_xml.exporter.**XML** (*exporter*)
Bases: object

Declaration type Model

Registry name Model.IO.Exporter.XML

Tablename io_exporter_xml

Inherit model or mixin

Blok Model Auth

class anyblok.bloks.model_authz.**ModelBasedAuthorizationBlok** (*registry*)
Bases: anyblok.blok.Blok

conditional_by = []

conflicting_by = []

classmethod import_declaration_module()

name = 'model_authz'

optional_by = []

classmethod reload_declaration_module (*reload*)

required_by = []

version = '0.9.6'

API doc

class anyblok.bloks.model_authz.models.**ModelPermissionGrant**
Bases: object

Default model for ModelBasedAuthorizationRule

Declaration type Model

Registry name Model.Authorization.ModelPermissionGrant

Tablename authorization_modelpermissiongrant

Inherit model or mixin

field name	Description
model	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
principal	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None
permission	<ul style="list-style-type: none"> •Context: •Label - None •primary_key - True •foreign_key - None •size - 64 •is crypted - False •Field type - <class 'any-blok.column.String'> •default - <class 'any-blok.column.NoDefaultValue'> •DB column name - None

0.9.6 (2017-07-03)

- [FIX] in One2Many and Many2Many field, the attribute model can be used on record node. Used for Polymorphisme

0.9.5 (2016-12-05)

- [ADD] Python 3.6 support
- Flake8

0.9.4 (2016-10-27)

- [FIX] Nose test pluggins load the configuration need for unit test
- [ADD] getFieldtype on SQLBase, this method return the type of the column

0.9.3 (2016-10-12)

- [FIX] SQLAlchemy 1.1.* add autoincrement='auto', or AnyBlok wait Boolean. If the field is an Integer and a primary_key with autoincrement='auto' then the value is True else False
- [FIX] SQLAlchemy 1.1.*, primary_key attribute don't define autoincrement. The column Integer with a primary_key=True without autoincrement declaration use autoincrement=True
- [FIX] SQLAlchemy 1.1.*, backref property check if the collection_class has __emulates__ attributes. InstrumentedList haven't to have this attribute

- [FIX] SQLAlchemy 1.1.*, Session State changed, update the update method of the registry to install / update / uninstall bloks
- [FIX] SQLAlchemy 1.1.*, Hybrid property don't propagate the relationship info attribute. The propagate is forced for Many2One and One2One. The only both relationships to be wrapped by hybrid_property
- [FIX] SQLAlchemy 1.1.*, Hybrid property wrap the fget result in the case of the fget is called on the class (not the instance). Adapt the unit test, don't check if the result id of column are the same, check if the expression give by this results are the same.
- [FIX] SQLAlchemy 1.1.*, listen can not be used with a hybrid_property. In the case of a listen, the mapper returned is not the hybrid_property but the real wrapped field

0.9.2 (2016-10-12)

- [FIX] setup.py: error with pip

0.9.1 (2016-10-3)

- [FIX] migration testcase
- [FIX] graphviz FORMATS
- [FIX] travis configuration

0.9.0 (2016-07-11)

- [REF] add Configuration.has method
- [FIX] test migration, force to load registry with unittest=True
- [FIX] test event
- [FIX] test blok
- [FIX] mapper with None parameter
- [FIX] add set_defaults in parser to update configuration dict
- [FIX] one2many remote columns
- [FIX] load anyblok.init in the unit test
- [IMP] Add plugins by configuration for:
 - Registry
 - Migration
 - get_url
- [IMP] add LogCapture
- [IMP] TestCase.Configuration, use to update Configuration only in a context manager
- [IMP] add Registry.db_exists class method, check with the configuration and the db_name if the connection is possible

0.8.5 (2016-06-20)

- [FIX] utf-8 encoding
- [REF] move bitbucket (mergural) to github (git)

0.8.4 (2016-06-14)

- [FIX] io/xml/importer one2many field
- [FIX] install blok, who are not in the blok list yet. But the blok is loaded

0.8.3 (2016-04-18)

- [FIX] cache and classmethod_cache on SQL model
- [ADD] is_installed classmethod cache

0.8.2 (2016-04-06)

- [REF] IO.Mapping methods delete and multi_delete can remove entry
- [FIX] datetime with timezone use timezone.localize, better than datetime.replace(tzinfo=...)
- [ADD] update sphinx extension

0.8.1 (2016-03-15)

- [FIX] [#21](#) Improve Decimal column setter
- [FIX] [#22](#) String, uString, Text and uText write ‘’ in database for False value
- [FIX] Change the external_id save in a two way
- [FIX] [#23](#) Column.Selection with None value, don’t return ‘None’ value by the getter

0.8.0 (2016-02-05)

Warning: Break the compatibility with the previous version of anyblok

- update method on the model replace

```
obj.update({field1: val1, ...})
```

by:

```
obj.update(field1=val1, ...)
```

- [REF] session expire is now on the attribute, the update method is refactored too.

- [FIX] blok: update version if the version change
- [REF] add required blok, this blok is installed and updated by the scripts anyblok_updatedb and anyblok_createdb
- [ADD] Add Color Column
- [REF] column can be encrypted
- [REF] DateTime column is not a naive datetime value
- [ADD] Add Password Column
- [ADD] Add UUID Column
- [ADD] Add URL Column

0.7.2 (2016-01-14)

- [FIX] delete flush after remove of the session
- [FIX] nose plugins
- [FIX] doesn't destroy automatically constraints (not created by anyblok), indexes (not created by anyblok), columns, tables by automigration, add options to force the delete of its.
- [REF] standardize the constraint and index names
- [FIX] Multi declaration of the same foreign key in the case of M2O and O2O
- [REF] SqlBase.update, become high level meth

0.7.1 (2016-01-08)

- [FIX] didn't cast the config data from the config file
- [IMP] copy init entry point from anyblok_pyramid

0.7.0 (2016-01-07)

Warning: Python 3.2 is not supported

- [REF] Add options to give database url, No break compatibility
- [REF] **the argument of ArgumentParser can be add in the configuration**
 - Improve the help of the application
 - Improve the type of the configuration, Work also with config file.
 - Adapt current configuration
- [REF] start to use sqlalchemy-utils, replace the database management
- [IMP] [#18](#) Forbidden the declaration of SQLAlchemy column or relationship
- [REF] [#15](#) Refactor unittest case to not create/drop database for each test

- [FIX] [#19](#) During migration if an unique constraint must be apply without unique value, then the constraint will be ignore and log a warning. No break the instalation of the blok
- [FIX] [#20](#) Update meth: expire the instance cause of relationship
- [IMP] refresh and expire meth on model
- [REF] delete obj, flush the session and delete the instance of obj of the session, before expire all the session, the goal is to reload the relation ship.
- [REF] [#13](#) Remove association model, replace it by call at the Blok definition
- [IMP] [#14](#) Add conflicting link between blok, two blok who are in conflict can be installed if the other is installed

0.6.0 (2016-01-07)

- [REF] unittest isolation
- [IMP] possibility to apply an extension for sqlalchemy
- [ADD] pool configuration

0.5.2 (2015-09-28)

- [IMP] extension for Sphinx and autodoc
- [ADD] API doc in doc
- [ADD] add foreign key option in relation ship
- [CRITICAL FIX] the EnvironnementManager didn't return the good scoped method for SQLAlchemy
- [CRITICAL FIX] the precommit_hook was not isolated by session
- [REF] add a named argument `must_be_loaded_by_unittest`, by default False, in `Configuration.add` to indicate if the function must be call during the initialisation of the unittest, generally for the configuration initialized by Environ variable

0.5.1 (2015-08-29)

- [IMP] unload declaration type callback

0.5.0 (2015-08-28)

Warning: Break the compatibility with the previous version of anyblok

- `cache`, `classmethod_cache`, `hybrid_method` and `listen` replace:

```
from anyblok import Declarations
cache = Declarations.cache
classmethod_cache = Declarations.classmethod_cache
hybrid_method = Declarations.hybrid_method
addListener = Declarations.addListener
```

by:

```
from anyblok.declarations import (cache, classmethod_cache,  
                                hybrid_method, listen)
```

Note: The listener can declare SQLAlchemy event

- declaration of the foreign key replace:

```
@register(Model):  
class MyClass:  
  
    myfield = Integer(foreign_key=(Model.System.Blok, 'name'))  
    myotherfield = Integer(foreign_key=('Model.System.Blok', 'name'))
```

by:

```
@register(Model):  
class MyClass:  
  
    myfield = Integer(foreign_key=Model.System.Blok.use('name'))  
    myotherfield = Integer(foreign_key="Model.System.Blok=>name")
```

- [IMP] add pop behaviour on **Model.System.Parameter**
- [REF] Load configuration before load blocks, to use Configuration during the declaration
- [FIX] all must return InstrumentedList, also when the result is empty
- [FIX] to_dict must not cast column
- [REF] add third entry in foreign key declaration to add options
- [IMP] ModelAttribute used to declare the need of specific attribute and get the attribute or the foreign key from this attribute
- [IMP] ModelAttributeAdapter, get a ModelAttribute from ModelAttribute or str
- [IMP] ModelRepr, Pseudo representation of a Model
- [IMP] ModelAdapter, get a ModelRepr from ModelRepr or str
- [IMP] ModelMapper and ModelAttributeMapper
- [REF] Event, the declaration of an event can be an anyblok or a sqlalchemy event
- [REF] the foreign key must be declared with ModelAttribute
- [REF] Use Adapter for Model and attribute in relation ship
- [REF] hybrid_method, cache and classmethod_cache are now only impotable decorator function
- [IMP] in column the default can be a classmethod name
- [REF] replace all the field (prefix, suffix, ...) by a formater field. It is a python formater string
- [IMP] Sequence column
- [IMP] add the default system or user configuration file

0.4.1 (2015-07-22)

Warning: Field Function change, fexp is required if you need filter

- [FIX] Field.Function, fexp is now a class method
- [REF] reduce flake8 complexity
- [REF] refactor field function
- [FIX] inherit relation ship from another model, thank Simon ANDRÉ for the bug report
- [REF] table/mapper args definition
- [REF] Refactor Field, Column, Relationship use now polymorphic inherit
- [FIX] Foreign key constraint, allow to add and drop constraint on more than one foreign key
- [ADD] update-all-bloks option
- [ADD] pre / post migration
- [REF] UML Diagram is now with autodoc script
- [REF] SQL Diagram is now with autodoc script
- [REF] Add **extend** key word in configuration file to extend an existing configuration

0.4.0 (2015-06-21)

Warning: Break the compatibility with the previous version of anyblok

- [REF] Add the possibility to add a logging file by argparse
- [ADD] No auto migration option
- [ADD] Plugin for nose to run unit test of the installed bloks
- [REF] The relation ship can be reference more than one foreign key
- [IMP] Add define_table/mapper_args methods to fill __table/mapper_args__ class attribute need to configure SQLAlchemy models
- [REF] Limit the commit in the registry only when the SQLA Session factory is recreated
- [REF] Commit and re-create the SQLA Session Factory, at installation, only if the number of Session inheritance of the number of Query inheritance change, else keep the same session
- [REF] Exception is not a Declarations type
- [FIX] Reload fonctionnality in python 3.2
- [REF] Remove the Declarations types Field, Column, Relationship, they are replaced by python import
- [REF] rename **ArgsParseManager** by **Configuration**
- [REF] rename **reload_module_if_blok_is_reloaded** by **reload_module_if_blok_is_reloading** method on blok
- [REF] rename **import_cfg_file** by **import_file** method on blok

- [REF] Consistency the argsparse configuration
- [REF] refactor part_to_load, the entry points loaded is bloks
- [IMP] Allow to define another column name in the table versus model
- [FIX] add importer for import configuration file
- [FIX] x2M importer without field just, external id

0.3.5 (2015-05-10)

- [IMP] When a new column is add, if the column have a default value, then this value will be added in all the entries where the value is null for this column
- [REF] import_cfg_file remove the importer when import has done

0.3.4 (2015-05-10)

- [ADD] logger.info on migration script to indicate what is changed
- [IMP] Add sequence facility in the declaration of Column
- [ADD] ADD XML Importer

0.3.3 (2015-05-04)

- [FIX] createdb script

0.3.2 (2015-05-04)

- [IMP] doc
- [REF] Use logging.config.configFile

0.3.1 (2015-05-04)

- [IMP] Update setup to add documentation files and blok's README

0.3.0 (2015-05-03)

- [IMP] Update Doc
- [FIX] Remove nullable column, the nullable constraint is removed not the column
- [ADD] Formater, convert value 2 str or str 2 value, with or without mapping
- [ADD] CSV Importer
- [REF] CSV Exporter to use Formater

0.2.12 (2015-04-29)

- [IMP] CSV Exporter
- [IMP] Exporter Model give external ID behaviour
- [ADD] Sequence model (Model.System.Sequence)
- [ADD] fields_description cached_classmethod with invalidation
- [ADD] Parameter Model (Model.System.Parameter)
- [FIX] environnement variable for test unitaire

0.2.11 (2015-04-26)

- [FIX] UNIT test createdb with prefix

0.2.10 (2015-04-26)

- [IMP] add enviroment variable for database information
- [ADD] argsparse option install all bloks
- [FIX] Python 3.2 need that bloks directory are python modules, add empty `__init__` file

0.2.9 (2015-04-18)

- [FIX] Add all rst at the main path of all the bloks

0.2.8 (2015-04-16)

- [IMP] unittest on SQLBase
- [IMP] add delete method on SQLBase to delete une entry from an instance of the model
- [REF] rename `get_primary_keys` to `get_mapping_primary_keys`, cause of `get_primary_keys` already exist in SQLBase

0.2.7 (2015-04-15)

- [IMP] Add IPython support for interpreter
- [REF] Update and Standardize the method to field the models (Field, Column, Relationship) now all the type of the column go on the ftype and comme from the name of the class

0.2.6 (2015-04-11)

- [FIX] use the backref name to get the label of the remote relation ship
- [FIX] add type information of the simple field

0.2.5 (2015-03-23)

- [FIX] In the parent / children relationship, where the pk is on a mixin or from inherit
- [FIX] How to Environment
- [FIX] Many2Many declared in Mixin
- [IMP] Many2One can now declared than the local column must be unique (only if the local column is not declared in the model)

0.2.3 (2015-03-23)

Warning: This version can be not compatible with the version **0.2.2**. Because in the foregn key model is a string you must replace the tablename by the registry name

- [FIX] Allow to add a relationship on the same model, the main use is to add parent / children relation ship on a model, They are any difference with the declaration of ta relation ship on another model
- [REF] standardize foreign_key and relation ship, if the str which replace the Model Declarations is now the registry name

0.2.2 (2015-03-15)

- [REF] **Unittest**
 - TestCase and DBTestCase are only used for framework
 - **BlokTestCase is used:**
 - * by `run_exit` function to test all the installed bloks
 - * at the installation of a blok if wanted

0.2.0 (2015-02-13)

Warning: This version is not compatible with the version **0.1.3**

- [REF] Import and reload are more explicite
- [ADD] **IO:**
 - Mapping: Link between Model instance and (Model, str key)

- [ADD] Env in registry_base to access at EnvironmentManager without to import it at each time
- [IMP] doc add how to on the environment

0.1.3 (2015-02-03)

- [FIX] setup long description, good for pypi but not for easy_install

0.1.2 (2015-02-02)

- [REFACTOR] Allow to declare Core components
- [ADD] Howto declare Core / Type
- [FIX] Model can only inherit simple python class, Mixin or Model
- [FIX] Mixin inherit chained
- [FIX] Flake8

0.1.1 (2015-01-23)

- [FIX] version, documentation, setup

0.1.0 (2015-01-23)

Main version of AnyBlok. You can with this version

- Create your own application
- Connect to a database
- Define bloks
- Install, Update, Uninstall the blok
- Define field types
- Define Column types
- Define Relationship types
- Define Core
- Define Mixin
- Define Model (SQL or not)
- Define SQL view
- Define more than one Model on a specific table
- Write unittest for your blok

Contents

- *ROADMAP*
 - *To implement*
 - *Library to include*
 - *Functionnnality which need a sprint*

To implement

- Add slogan
- Update doc
- Need improve alembic, sqlalchemy-util
- Refactor the engine declarations to have master / slave(s) configuration
- Addons for sqlalchemy : <http://sqlalchemy-utils.readthedocs.org/en/latest/installation.html>

Library to include

- full text search: <https://pypi.python.org/pypi/SQLAlchemy-FullText-Search/0.2>
- internationalisation: <https://pypi.python.org/pypi/SQLAlchemy-i18n/0.8.2>
- sqltap <http://sqltap.inconshreveable.com>, profiling and introspection for SQLAlchemy applications
- Crypt <https://bitbucket.org/zzzeek/sqlalchemy/wiki/UsageRecipes/DatabaseCrypt>
- profiling <https://bitbucket.org/zzzeek/sqlalchemy/wiki/UsageRecipes/Profiling>

Functionnality which need a sprint

- Tasks Management
- Internalization
- Ancestor left / right

Contents

- *Mozilla Public License Version 2.0*
 - *1. Definitions*
 - * *1.1. “Contributor”*
 - * *1.2. “Contributor Version”*
 - * *1.3. “Contribution”*
 - * *1.4. “Covered Software”*
 - * *1.5. “Incompatible With Secondary Licenses”*
 - * *1.6. “Executable Form”*
 - * *1.7. “Larger Work”*
 - * *1.8. “License”*
 - * *1.9. “Licensable”*
 - * *1.10. “Modifications”*
 - * *1.11. “Patent Claims” of a Contributor*
 - * *1.12. “Secondary License”*
 - * *1.13. “Source Code Form”*
 - * *1.14. “You” (or “Your”)*
 - *2. License Grants and Conditions*
 - * *2.1. Grants*
 - * *2.2. Effective Date*
 - * *2.3. Limitations on Grant Scope*
 - * *2.4. Subsequent Licenses*
 - * *2.5. Representation*
 - * *2.6. Fair Use*
 - * *2.7. Conditions*
 - *3. Responsibilities*
 - * *3.1. Distribution of Source Form*
 - * *3.2. Distribution of Executable Form*
 - * *3.3. Distribution of a Larger Work*
 - * *3.4. Notices*
 - * *3.5. Application of Additional Terms*
 - *4. Inability to Comply Due to Statute or Regulation*
 - *5. Termination*
 - * *5.1.*
 - * *5.2.*

- * 5.3.
- 6. *Disclaimer of Warranty*
- 7. *Limitation of Liability*
- 8. *Litigation*
- 9. *Miscellaneous*
- 10. *Versions of the License*
 - * 10.1. *New Versions*
 - * 10.2. *Effect of New Versions*
 - * 10.3. *Modified Versions*
 - * 10.4. *Distributing Source Code Form that is Incompatible With Secondary Licenses*
- *Exhibit A - Source Code Form License Notice*
- *Exhibit B - “Incompatible With Secondary Licenses” Notice*

1. Definitions

1.1. “Contributor”

Means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.

1.2. “Contributor Version”

Means the combination of the Contributions of others (if any) used by a Contributor and that particular Contributor’s Contribution.

1.3. “Contribution”

Means Covered Software of a particular Contributor.

1.4. “Covered Software”

Means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof.

1.5. “Incompatible With Secondary Licenses”

Means:

- **That the initial Contributor has attached the notice described in Exhibit B** to the Covered Software; or
- **That the Covered Software was made available under the terms of version 1.1** or earlier of the License, but not also under the terms of a Secondary License.

1.6. “Executable Form”

Means any form of the work other than Source Code Form.

1.7. “Larger Work”

Means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.

1.8. “License”

Means this document.

1.9. “Licensable”

Means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.

1.10. “Modifications”

Means any of the following:

- **Any file in Source Code Form that results from an addition to, deletion from,** or modification of the contents of Covered Software; or
- Any new file in Source Code Form that contains any Covered Software.

1.11. “Patent Claims” of a Contributor

Means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.

1.12. “Secondary License”

Means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.

1.13. “Source Code Form”

Means the form of the work preferred for making modifications.

1.14. “You” (or “Your”)

Means an individual or a legal entity exercising rights under this License. For legal entities, “You” includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants and Conditions

2.1. Grants

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

- **Under intellectual property rights (other than patent or trademark)** Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and
- **Under Patent Claims of such Contributor to make, use, sell, offer for sale,** have made, import, and otherwise transfer either its Contributions or its Contributor Version.

2.2. Effective Date

The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.

2.3. Limitations on Grant Scope

The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:

- For any code that a Contributor has removed from Covered Software; or
- **For infringements caused by: (i) Your and any other third party's** modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or
- **Under Patent Claims infringed by Covered Software in the absence of its** Contributions.

This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).

2.4. Subsequent Licenses

No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).

2.5. Representation

Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.

2.7. Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

3. Responsibilities

3.1. Distribution of Source Form

All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipients' rights in the Source Code Form.

3.2. Distribution of Executable Form

If You distribute Covered Software in Executable Form then:

- **Such Covered Software must also be made available in Source Code Form, as** described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and
- **You may distribute such Executable Form under the terms of this License, or** sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipients' rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4. Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Termination

5.1.

The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated (a) provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2.

If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3.

In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

6. Disclaimer of Warranty

Warning: Covered Software is provided under this License on an “as is” basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

7. Limitation of Liability

Warning: Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such party's negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

8. Litigation

Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a party's ability to bring cross-claims or counter-claims.

9. Miscellaneous

This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

10. Versions of the License

10.1. New Versions

Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.

10.2. Effect of New Versions

You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.

10.3. Modified Versions

If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the license and remove any references to the name of the license steward (except to note that such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses

If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice

This Source Code Form **is** subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was **not** distributed **with** this file, You can obtain one at <http://mozilla.org/MPL/2.0/>.

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.

Note: You may add additional accurate notices of copyright ownership.

Exhibit B - “Incompatible With Secondary Licenses” Notice

This Source Code Form is “Incompatible With Secondary Licenses”, as defined by the Mozilla Public License, v. 2.0.

CHAPTER 12

Indices and tables

- `genindex`
- `modindex`
- `search`

a

anyblok, 53
anyblok._graphviz, 126
anyblok.blok, 68
anyblok.bloks.anyblok_core, 139
anyblok.bloks.anyblok_core.authorization, 141
anyblok.bloks.anyblok_core.core.base, 141
anyblok.bloks.anyblok_core.core.instrumentedlist, 145
anyblok.bloks.anyblok_core.core.query, 146
anyblok.bloks.anyblok_core.core.session, 146
anyblok.bloks.anyblok_core.core.sqlbase, 142
anyblok.bloks.anyblok_core.core.sqlviewbase, 145
anyblok.bloks.anyblok_core.documentation, 163
anyblok.bloks.anyblok_core.documentation.blok, 163
anyblok.bloks.anyblok_core.documentation.model, 163
anyblok.bloks.anyblok_core.documentation.model.attribute, 163
anyblok.bloks.anyblok_core.documentation.model.field, 164
anyblok.bloks.anyblok_core.exceptions, 164
anyblok.bloks.anyblok_core.system, 146
anyblok.bloks.anyblok_core.system.blok, 146
anyblok.bloks.anyblok_core.system.cache, 148
anyblok.bloks.anyblok_core.system.column, 152
anyblok.bloks.anyblok_core.system.cron, 156
anyblok.bloks.anyblok_core.system.field, 150
anyblok.bloks.anyblok_core.system.model, 158
anyblok.bloks.anyblok_core.system.parameter, 159
anyblok.bloks.anyblok_core.system.relationship, 154
anyblok.bloks.anyblok_core.system.sequence, 161
anyblok.bloks.io, 164
anyblok.bloks.io.exceptions, 166
anyblok.bloks.io.exporter, 172
anyblok.bloks.io.formater, 172
anyblok.bloks.io.importer, 170
anyblok.bloks.io.mapping, 167
anyblok.bloks.io.mixin, 169
anyblok.bloks.io_csv, 175
anyblok.bloks.io_csv.exceptions, 177
anyblok.bloks.io_csv.exporter, 181
anyblok.bloks.io_csv.importer, 179
anyblok.bloks.io_csv.mixin, 177
anyblok.bloks.io_xml, 183
anyblok.bloks.io_xml.exceptions, 186
anyblok.bloks.io_xml.exporter, 187
anyblok.bloks.io_xml.importer, 186
anyblok.bloks.model_authz, 187
anyblok.bloks.model_authz.models, 187
anyblok.column, 85
anyblok.config, 61
anyblok.declarations, 53
anyblok.environment, 66
anyblok.field, 82
anyblok.imp, 66
anyblok.logging, 64
anyblok.mapper, 59
anyblok.migration, 77
anyblok.model, 58
anyblok.registry, 70

`anyblok.relationship`, [114](#)
`anyblok.scripts`, [128](#)
`anyblok.tests.testcase`, [131](#)

A

anyblok (module), 53
 anyblok._graphviz (module), 126
 anyblok.blok (module), 68
 anyblok.bloks.anyblok_core (module), 139
 anyblok.bloks.anyblok_core.authorization (module), 141
 anyblok.bloks.anyblok_core.core.base (module), 141
 anyblok.bloks.anyblok_core.core.instrumentedlist (module), 145
 anyblok.bloks.anyblok_core.core.query (module), 146
 anyblok.bloks.anyblok_core.core.session (module), 146
 anyblok.bloks.anyblok_core.core.sqlbase (module), 142
 anyblok.bloks.anyblok_core.core.sqlviewbase (module), 145
 anyblok.bloks.anyblok_core.documentation (module), 163
 anyblok.bloks.anyblok_core.documentation.blok (module), 163
 anyblok.bloks.anyblok_core.documentation.model (module), 163
 anyblok.bloks.anyblok_core.documentation.model.attribute (module), 163
 anyblok.bloks.anyblok_core.documentation.model.field (module), 164
 anyblok.bloks.anyblok_core.exceptions (module), 164
 anyblok.bloks.anyblok_core.system (module), 146
 anyblok.bloks.anyblok_core.system.blok (module), 146
 anyblok.bloks.anyblok_core.system.cache (module), 148
 anyblok.bloks.anyblok_core.system.column (module), 152
 anyblok.bloks.anyblok_core.system.cron (module), 156
 anyblok.bloks.anyblok_core.system.field (module), 150
 anyblok.bloks.anyblok_core.system.model (module), 158
 anyblok.bloks.anyblok_core.system.parameter (module), 159
 anyblok.bloks.anyblok_core.system.relationship (module), 154
 anyblok.bloks.anyblok_core.system.sequence (module), 161
 anyblok.bloks.io (module), 164
 anyblok.bloks.io.exceptions (module), 166
 anyblok.bloks.io.exporter (module), 172
 anyblok.bloks.io.formater (module), 172
 anyblok.bloks.io.importer (module), 170
 anyblok.bloks.io.mapping (module), 167
 anyblok.bloks.io.mixin (module), 169
 anyblok.bloks.io_csv (module), 175
 anyblok.bloks.io_csv.exceptions (module), 177
 anyblok.bloks.io_csv.exporter (module), 181
 anyblok.bloks.io_csv.importer (module), 179
 anyblok.bloks.io_csv.mixin (module), 177
 anyblok.bloks.io_xml (module), 183
 anyblok.bloks.io_xml.exceptions (module), 186
 anyblok.bloks.io_xml.exporter (module), 187
 anyblok.bloks.io_xml.importer (module), 186
 anyblok.bloks.model_authz (module), 187
 anyblok.bloks.model_authz.models (module), 187
 anyblok.column (module), 85
 anyblok.config (module), 61
 anyblok.declarations (module), 53
 anyblok.environment (module), 66
 anyblok.field (module), 82
 anyblok.imp (module), 66
 anyblok.logging (module), 64
 anyblok.mapper (module), 59
 anyblok.migration (module), 77
 anyblok.model (module), 58
 anyblok.registry (module), 70
 anyblok.relationship (module), 114
 anyblok.scripts (module), 128
 anyblok.tests.testcase (module), 131