AnyBlok Documentation

Release 0.2.4

Jean-Sebastien SUZANNE

June 26, 2016

Contents

1 Front Matter **3**
- 1.1 Project Homepage . . . 3
- 1.2 Project Status . . . 3
- 1.3 Installation . . . 3
- 1.4 Unit Test . . . 3
- 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 Create a Blok group . . . 7
- 2.2 Create Bloks . . . 8
- 2.3 Create Models . . . 11
- 2.4 Updating an existing Model . . . 13
- 2.5 Add entries in the argsparse configuration . . . 14
- 2.6 Create an application . . . 14
- 2.7 Create an interpreter . . . 18

3 How to add a new `Type` /core **21**
- 3.1 Difference between `Core` and `Type` . . . 21
- 3.2 Declare a new `Type` . . . 21
- 3.3 Declare a Mixin entry type . . . 23
- 3.4 Declare a new `Core` . . . 23

4 Environmment **25**
- 4.1 Use the current environment . . . 25
- 4.2 Define a new environment type . . . 26

5 IO **27**
- 5.1 Mapping . . . 27

6 MEMENTO **29**
- 6.1 Blok . . . 29
- 6.2 Declaration . . . 30
- 6.3 Model . . . 31
- 6.4 Column . . . 34
- 6.5 RelationShip . . . 36

6.6 Field . . . 37
6.7 Mixin . . . 37
6.8 SQL View . . . 38
6.9 Core . . . 39
6.10 Sharing a table between more than one model . . . 40
6.11 Sharing a view between more than one model . . . 41
6.12 Specific behaviour . . . 41

7 AnyBlok framework **47**
7.1 anyblok module . . . 47
7.2 anyblok.declarations module . . . 47
7.3 anyblok._argsparse module . . . 54
7.4 anyblok._imp module . . . 57
7.5 anyblok._logging module . . . 57
7.6 anyblok.environment module . . . 59
7.7 anyblok.blok module . . . 60
7.8 anyblok.registry module . . . 61
7.9 anyblok.migration module . . . 66
7.10 anyblok._graphviz module . . . 71
7.11 anyblok.databases module . . . 74
7.12 anyblok.scripts module . . . 74

8 Helper for unittest **77**
8.1 TestCase . . . 77
8.2 DBTestCase . . . 78
8.3 BlokTestCase . . . 79

9 Bloks **81**
9.1 anyblok-core blok . . . 81

10 CHANGELOG **83**
10.1 Future . . . 83
10.2 0.2.3 . . . 83
10.3 0.2.2 . . . 83
10.4 0.2.0 . . . 84
10.5 0.1.3 . . . 84
10.6 0.1.2 . . . 84
10.7 0.1.1 . . . 84
10.8 0.1.0 . . . 84

11 ROADMAP **87**
11.1 Next step for the 0.2 . . . 87
11.2 To implement . . . 87
11.3 Library to include . . . 87
11.4 Functionnality which need a sprint . . . 87

12 Mozilla Public License Version 2.0 **91**
12.1 1. Definitions . . . 91
12.2 2. License Grants and Conditions . . . 93
12.3 3. Responsibilities . . . 94
12.4 4. Inability to Comply Due to Statute or Regulation . . . 95
12.5 5. Termination . . . 95
12.6 6. Disclaimer of Warranty . . . 95
12.7 7. Limitation of Liability . . . 96
12.8 8. Litigation . . . 96

12.9 9. Miscellaneous 96
12.10 10. Versions of the License 96
12.11 Exhibit A - Source Code Form License Notice 97
12.12 Exhibit B - “Incompatible With Secondary Licenses” Notice 97

13 Indices and tables **99**

Python Module Index **101**

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*

CHAPTER 1

Front Matter

Information about the AnyBlok project.

1.1 Project Homepage

AnyBlok is hosted on Bitbucket - the main project page is at https://bitbucket.org/jssuzanne/anyblok or http://code.anyblok.org. Source code is tracked here using Mercurial.

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.

1.2 Project Status

AnyBlok is currently in beta 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 http://code.anyblok.org/get/default.tar.gz. or http://code.anyblok.org/get/default.zip

1.3 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.

1.4 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
```

1.5 Dependencies

AnyBlok works with **Python 3.2** and later. The install process will ensure that SQLAlchemy, Alembic are installed, in addition to other dependencies. AnyBlok will work with SQLAlchemy as of version **0.9.8**. AnyBlok will work with Alembic as of version **0.7.3**. The latest version of them is strongly recommended.

1.6 Contributing (hackers needed!)

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

1.7 Author

Jean-Sébastien Suzanne

1.8 Contributors

Anybox team:

- Georges Racinet
- Christophe Combelles
- Sandrine Chaufournais
- Jean-Sébastien Suzanne
- Florent Jouatte
- Simon André
- Clovis Nzouendjou
- Pierre Verkest
- Franck Bret

1.9 Bugs

Bugs and feature enhancements to AnyBlok should be reported on the Issue tracker.

Contents

- *How to create your own application*
  - *Create a Blok group*
  - *Create Bloks*
  - *Create Models*
  - *Updating an existing Model*
  - *Add entries in the argsparse configuration*
  - *Create an application*
  - *Create an interpreter*

CHAPTER 2

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 functionnalities.

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

2.1 Create a Blok group

A blok group is a `setuptools` entry point. Splitting the project into several groups allows to select the bloks needed by the application. This separation also allows a blok to come from more than one blok group: it is not the same blok but they have the same name. You can provide two implementations for the same thing and use the right implementation depending on the context.

For this example, the blok group `WorkBlok` will be used

File tree:

```
WorkBlok
-- setup.py
```

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

```
WorkBlok = [
    '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={
        'WorkBlok': WorkBlok,
    },
)
```

2.2 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 will be imported by anyblok.

Warning: Modules and packages starting with _ are not imported, the package tests are also not imported.

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
-- argsparse.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 Chaufournais',
                                      '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 argsparse  # 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.

2.3 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
register = Declarations.register
Model = Declarations.Model
String = Declarations.Column.String


@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
register = Declarations.register
Model = Declarations.Model
Integer = Declarations.Column.Integer
String = Declarations.Column.String
Many2One = Declarations.RelationShip.Many2One


@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
register = Declarations.register
Model = Declarations.Model
String = Declarations.Column.String


@register(Model)
class Position:

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

    def __str__(self):
        return self.name
```

employee_blok.employee:

```
from anyblok import Declarations
register = Declarations.register
Model = Declarations.Model
String = Declarations.Column.String
Many2One = Declarations.RelationShip.Many2One


@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)
```

2.4 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
register = Declarations.register
Model = Declarations.Model
Many2One = Declarations.RelationShip.Many2One


@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)
```

2.5 Add entries in the argsparse configuration

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

employee_blok.arsparse:

```
from anyblok._argsparse import ArgsParseManager


@ArgsParseManager.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')
```

2.6 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'],
        'WorkBlok': WorkBlok,
    },
)
```

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._argsparse import ArgsParseManager
```

```
logger = getLogger(__name__)


def exampleblok():
    # Initialise the application, with a name and a version number
    # select the groupe of options to display
    # select the groups of bloks availlable
    # return a registry if the database are selected
    registry = anyblok.start(
        'Example Blok', '1.0',
        argsparse_groups=['config', 'database', 'message'],
        parts_to_load=['AnyBlok', 'WorkBlok'])

    if not registry:
        return

    message_before = ArgsParseManager.get('message_before')
    message_after = ArgsParseManager.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] [-c CONFIGFILE] [--message-before MESSAGE_BEFORE]
                   [--message-after MESSAGE_AFTER] [--db_name DBNAME]
                   [--db_drivername DBDRIVERNAME] [--db_username DBUSERNAME]
                   [--db_password DBPASSWORD] [--db_host DBHOST]
                   [--db_port DBPORT]

Example Blok - 1.0

optional arguments:
    -h, --help            show this help message and exit
    -c CONFIGFILE         Relative path of the config file

This is the 'message' group:
    --message-before MESSAGE_BEFORE
    --message-after MESSAGE_AFTER

Database:
    --db_name DBNAME      Name of the database
    --db_drivername DBDRIVERNAME
                          the name of the database backend. This name will
                          correspond to a module in sqlalchemy/databases or a
                          third party plug-in
    --db_username DBUSERNAME
The user name
    --db_password DBPASSWORD
```

```
database password
    --db_host DBHOST      The name of the host
    --db_port DBPORT      The port number
```

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 ..."
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_s
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 b
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' a
2014-1129 10:54:27 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named 'system_cache_id_s
2014-1129 10:54:27 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named 'room_id_seq' as c
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 b
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' a
2014-1129 10:54:28 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named 'system_cache_id_s
2014-1129 10:54:28 INFO - anyblok:alembic.ddl.postgresql - Detected sequence named 'room_id_seq' as c
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 b
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_s
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 b
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_s
2014-1129 10:54:29 INFO - anyblok:alembic.autogenerate.compare - Detected added column 'employee.posi
2014-1129 10:54:29 WARNING - anyblok:anyblok.migration - (IntegrityError) column "position_name" cont
'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 b
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:54:30 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Get the employee ...
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Sandrine Chaufournais in Room 308 at 14-16 ru
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Christophe Combelles in Room 308 at 14-16 rue
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Clovis Nzouendjou in Room 308 at 14-16 rue So
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Florent Jouatte in Room 308 at 14-16 rue Sole
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Simon André in Room 308 at 14-16 rue Soleille
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Jean-Sébastien Suzanne in Room 308 at 14-16 r
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Georges Racinet in Room 308 at 14-16 rue Sole
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Pierre Verkest in Room 308 at 14-16 rue Solei
2014-1129 10:54:30 INFO - anyblok:exampleblok.scripts - Franck Bret in Room 308 at 14-16 rue Soleille
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 ...'
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_s
2014-1129 10:57:52 INFO - anyblok:alembic.autogenerate.compare - Detected NOT NULL on column 'employe
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
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:57:52 INFO - anyblok:anyblok.bloks.anyblok_core.declarations.system.blok - Load the blok
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Get the employee ...
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Sandrine Chaufournais in Room 308 at 14-16 ru
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Christophe Combelles in Room 308 at 14-16 rue
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Clovis Nzouendjou in Room 308 at 14-16 rue So
```

```
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Florent Jouatte in Room 308 at 14-16 rue Sole
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Simon André in Room 308 at 14-16 rue Soleille
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Jean-Sébastien Suzanne in Room 308 at 14-16 r
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Georges Racinet in Room 308 at 14-16 rue Sole
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Pierre Verkest in Room 308 at 14-16 rue Solei
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - Franck Bret in Room 308 at 14-16 rue Soleille
2014-1129 10:57:52 INFO - anyblok:exampleblok.scripts - End ...
```

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

2.7 Create an interpreter

Anyblok provides some functions to help creating an application:

- createdb
- updatedb
- interpreter
- run_exit (nose test)

Here is how to create an interpreter:

```
from anyblok.scripts import interpreter


def exampleblok_interpreter():
    interpreter(
        'Interpreter', '1.0',
        argsparse_groups=['config', 'database', 'interpreter'],
        parts_to_load=['AnyBlok', 'WorkBlok'])
```

```
jssuzanne:anyblok jssuzanne$ ./bin/exampleblok_interpreter -c anyblok.cfg
2014-0428 20:57:38 INFO - anyblok:root - Registry.load
2014-0428 20:57:38 INFO - anyblok:anyblok.registry - Blok 'anyblok-core' loaded
2014-0428 20:57:38 INFO - anyblok:anyblok.registry - Blok 'office' loaded
2014-0428 20:57:38 INFO - anyblok:anyblok.registry - Blok 'position' loaded
2014-0428 20:57:38 INFO - anyblok:anyblok.registry - Blok 'employee' loaded
2014-0428 20:57:38 INFO - anyblok:anyblok.registry - Blok 'employee-position' loaded
2014-0428 20:57:38 INFO - anyblok:anyblok.registry - Assemble 'Model' entry
2014-0428 20:57:39 INFO - anyblok:alembic.migration - Context impl PostgresqlImpl.
2014-0428 20:57:39 INFO - anyblok:alembic.migration - Will assume transactional DDL.
2014-0428 20:57:39 INFO - anyblok:anyblok.registry - Initialize 'Model' entry
Python 3.3.5 (default, Mar 12 2014, 15:18:42)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> [emp.name for emp in registry.Employee.query()]
['Clovis Nzouendjou', 'Franck Bret', 'Florent Jouatte', 'Georges Racinet',
 'Sandrine Chaufournais', 'Simon André', 'Pierre Verkest',
 'Jean-Sébastien Suzanne', 'Christophe Combelles']
```

TODO: I know it's not a setuptools documentation but it could be kind to show a complete minimalist exampe 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.

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`*

CHAPTER 3

How to add a new `Type` /core

`Type` and `Core` are both `Declarations`.

3.1 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

3.2 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
- Field
- Exception

Some are directly see as a database declaration (entry `Type`):

- Model
- Mixin
- Core

Other have not depends with the installation of not of the bloks:

- Field
- Exception

> **Warning:** All declaration of new entry must be done out of any bloks

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 `Declarations.register`
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 `Type` is an entry, depend of the installation or not of the bloks

assemble	**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:
	`@add_declaration_type(isAnEntry=True, assemble='assemble')`
	`class MyTpe:`
	`    ...`
	`    @classmethod`
	`    def assemble(cls, registry):`
	`        ...`
	Warning: registry is the registry of the database

initialize	**Only for the entry "Type"** Waiting the name of the classmethod which make the action to initialize the registry:
	`@add_declaration_type(isAnEntry=True, initialize='initialize')`
	`class MyTpe:`
	`    ...`
	`    @classmethod`
	`    def initialize(cls, registry):`
	`        ...`
	Warning: registry is the registry of the database

3.3 Declare a Mixin entry type

`Mixin` is a `Type` to add behaviours, it is not a particular `Type`. But it is always very interresting 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
```

3.4 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

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

CHAPTER 4

Environmment

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

4.1 Use the current environment

The environment can be used whereever in the code.

4.1.1 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)
```

4.1.2 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)
```

4.2 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

- *IO*
  - *Mapping*

CHAPTER 5

IO

Note: Require tha the anyblok-io is installed

TODO

5.1 Mapping

`Model.IO.Mapping` allows to link a `Model` instance by a `Model` namesapce 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 `IOMapingSetException` will be raised. Il 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
```

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*
    - *Cache*
    - *Event*
    - *Hybrid method*
    - *Pre-commit hook*
    - *Aliased*
    - *Get the registry*
    - *Get the current environment*

CHAPTER 6

MEMENTO

Anyblok mainly depends on:

- Python 3.2+
- SQLAlchemy
- Alembic

6.1 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
`__doc__`	Short description of the blok (in the docstring)
`version`	the version of the blok (required because no value by default)
`autoinstall`	boolean, if `True` this blok is automatically installed
`priority`	installation order of the blok to installation
`readme`	Path of the 'readme' file of the blok, by default `README.rst`

And the methods that define blok behaviours:

Method	Description
`import_declaration_module`	classmethod, call to import all python module which declare object from blok.
`reload_declaration_module`	classmethod, call to reload the import all the python module which declare object
`update`	Action to do when the blok is being install or updated. This method has one argument `latest_version` (None for install)
`uninstall`	Action to do when the blok is being uninstalled
`load`	Action to do when the server starts

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={
        'AnyBlok': [
            'web=anyblok_web_server.bloks.web:Web',
        ],
    },
    ...
)
```

6.2 Declaration

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

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

The `Declarations` has two main methods

Method name	Description
`register`	Add the declaration in the registry This method can be used as: • A function: `class Foo:` `  pass` `register(``Declarations.type``, cls_=Foo)` • A decorator: `@register(``Declarations.type``)` `class Foo:` ` pass`

`unregister`	Remove an existing declaration from the registry. This method is only used as a function:
	`from ... import Foo`
	`unregister(``Declarations.type``, cls_=Foo)`

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

- Model
- Column
- ...

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

6.3 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 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 exist 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 `register` is used as a function not as a decorator

name_	Overload the name of the class:
	`@register(Model, name_='Bar')`
	`class Foo:`
	`  pass`
	
	`Declarations.Bar`

tablename	Overload the name of the table:
	`@register(Model, tablename='my_table')`
	`class Foo:`
	`  pass`

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:
	• str
	`@register(Model, tablename='foo')`
	`class Bar:`
	`  pass`
	• declaration
	`@register(Model, tablename=Model.Foo)`
	`class Bar:`
	`  pass`

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

6.3.1 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
```

6.3.2 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

6.3.3 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.

6.4 Column

To declare a `Column` in a model, add a column on the table of the model. All the column type are in the `Declarations`:

```
from anyblok.declarations import Declarations

Integer = Declarations.Column.Integer
String = Declarations.Column.String


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

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

List of the `Declarations` of the column type:

- `DateTime`: use datetime.datetime
- `Decimal`: use decimal.Decimal
- `Float`
- `Time`: use datetime.time
- `BigInteger`
- `Boolean`
- `Date`: use datetime.date
- `Integer`

- `Interval`: use the datetime.timedelta
- `LargeBinary`
- `SmallInteger`
- `String`
- `Text`
- `uString`
- `uText`
- `Selection`
- `Json`

All the columns have the following parameters:

Parameter	Description
label	Label of the column, If None the label is the name of column capitalized
default	define a default value for this column. ..warning:: the default value depends of the column type
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	Define a foreign key on this column to another column of another model:
	`@register(Model)`
	`class Foo:`
	`    id : Integer(primary_key=True)`
	
	`@register(Model)`
	`class Bar:`
	`    id : Integer(primary_key=True)`
	`    foo: Integer(foreign_key=(Model.Foo, 'id'))`
	If the `Model` Declarations doesn't exist yet, you can use the regisrty name:
	`foo: Integer(foreign_key=('Model.Foo', 'id'))`

Other attribute for `String` and `uString`:

Param	Description
`size`	Column size in the bdd

Other attribute for `Selection`:

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

6.5 RelationShip

To declare a `RelationShip` in a model, add a RelationShip on the table of the model. All the RelationShip types are in the `Declarations`:

```
from anyblok.declarations import Declarations

Integer = Declarations.Column.Integer
Many2One = Declarations.RelationShip.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)
```

List of the `Declarations` 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_colum`	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 relationship
`nullable`	Indicates if the column name is nullable or not
`backref`	Remote One2One link with the column name

Parameters of the `Many2One` field:

Parame-ter	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 relation ship
`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_column	Column name in the join table which have got the foreign key to the remote model
local_column	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_column	Column name in the join table which holds the foreign key to the model
many2many	Opposite Many2Many link with this relationship

6.6 Field

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

```
from anyblok.declarations import Declarations

Integer = Declarations.Column.Integer
Fuction = Declarations.Field.Function


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

    id = Integer(primary_key=True)
    myid = Function(fget='get_my_id')

    def get_my_id(self):
        return self.id
```

List of the Declarations of the Field type:

- Function

Parameters for Field.Function

Parameter	Description

fget	name of the method to call to get the value of field:
	`def fget(self):`
	`    return self.id`

model	The remote model
remote_column	The column name on the remote model, if no remote columns are given, the remote column will be the primary column of the remote model

6.7 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')
```

6.8 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)
```

6.9 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.

6.9.1 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
```

6.9.2 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
```

6.9.3 SqlViewBase

Like `SqlBase`, only the `SqlView` inherits this `Core` class.

```
from anyblok import Declarations


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

6.9.4 Query

Overloads the SQLAlchemy `Query` class.

```
from anyblok import Declarations


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

6.9.5 Session

Overloads the SQLAlchemy `Session` class.

```
from anyblok import Declarations


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

6.9.6 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)
```

6.10 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.

6.11 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

6.12 Specific behaviour

AnyBlok implements some facilities to help developers

6.12.1 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 `Declarations.cache` or `Declarations.classmethod_cache` to apply a cache on a method

Warning: `Declarations.cache` depend of the instance, if you want add a cache for any instance you must use `Declarations.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()
```

6.12.2 Event

Simple implementation of a synchronous `event`:

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

@register(Model)
class Test:

        x = 0

        @Declarations.addListener(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 `addListener` which binds the decorated method to the event.
- **`fire` method with the following parameters:**
  - `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 `addListener` decorator but the classmethod decorator, because the method name is already seen as an event listener

6.12.3 Hybrid method

Facility to create an SQLAlchemy hybrid method. See this page: http://docs.sqlalchemy.org/en/latest/orm/extensions/hybrid.html#modul 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:

```
@register(Model)
class Test:

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

6.12.4 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')
```

6.12.5 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 functionnality 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

6.12.6 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'
```

6.12.7 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', defaul="My default value")
```

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

Contents

- *AnyBlok framework*
  - *anyblok module*
  - *anyblok.declarations module*
  - *anyblok._argsparse module*
  - *anyblok._imp module*
  - *anyblok._logging module*
  - *anyblok.environment module*
  - *anyblok.blok module*
  - *anyblok.registry module*
  - *anyblok.migration module*
  - *anyblok._graphviz module*
  - *anyblok.databases module*
    - *anyblok.databases.postgres module*
  - *anyblok.scripts module*

CHAPTER 7

AnyBlok framework

7.1 anyblok module

anyblok.**start**(*processName*, *version='0.2.4'*, *prompt='%(processName)s - %(version)s'*, *argsparse_groups=None*, *parts_to_load=None*, *logger=None*, *useseparator=False*)

Function which initialize the application

```
registry = start('My application',
                 argsparse_groups=['config', 'database'],
                 parts_to_load=['AnyBlok'])
```

Parameters

- **processName** – Name of the application
- **version** – Version of the application
- **prompt** – Prompt message for the help
- **argsparse_groups** – list of the group of option for argparse
- **parts_to_load** – group of blok to load
- **logger** – option to configure logging
- **useseparator** – boolean, indicate if argsparse option are split betwwen two application

Return type registry if the database name is in the configuration

7.2 anyblok.declarations module

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 Column(**args*, ***kwargs*)

Column class

This class can't be instanciated

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

get_tablename(*registry*, *model*)
Return the table name of the remote model

Return type str of the table name

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

native_type()
Return the native SqlAlchemy type

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.**Exception**
Adapter to Exception Class

The Exception class is used to define the type of Declarations Exception

Add new Exception type:

```
@Declarations.register(Declarations.Exception)
class MyException:
    pass
```

Removing the exception is forbidden because it can be used

exception ArgsParseManagerException
Simple Exception for ArgsParseManager

exception Declarations.Exception.**BlokManagerException**(**args*, ***kwargs*)
Simple exception to BlokManager

exception Declarations.Exception.**DeclarationsException**
Simple Exception for Declarations

exception Declarations.Exception.**EnvironmentException**
Exception for the Environment

exception Declarations.Exception.**FieldException**
Simple Exception for Field

exception Declarations.Exception.**ImportManagerException**
Exception for Import Manager

exception Declarations.Exception.**MigrationException**
Simple Exception class for Migration

exception Declarations.Exception.**ModelException**
Exception for Model declaration

exception Declarations.Exception.**RegistryException**
Simple Exception for Registry

exception Declarations.Exception.**RegistryManagerException**
Simple Exception for Registry

exception Declarations.Exception.**ViewException**
Exception for View declaration

classmethod Declarations.Exception.**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 to add in the declaration

Exception DeclarationsException

classmethod Declarations.Exception.**unregister**(*entry*, *cls_*)
Forbidden method
Exception DeclarationsException

class Declarations.**Field**(**args*, ***kwargs*)
Field class

This class can't be instanciated

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_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

native_type()
Return the native SqlAlchemy type

Exception FieldException

classmethod register(*parent*, *name*, *cls_*, ***kwargs*)
add new sub registry in the registry

Parameters

- **parent** – Parent to attach the declaration to
- **name** – Name of the new field
- **cls** – Class to add in the declaration

Exception FieldException

classmethod unregister(*child*, *cls_*)
Forbidden method

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

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_listner**(*registry*, *namespace*, *base*, *properties*)

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

classmethod **apply_hybrid_method**(*registry*, *namespace*, *bases*, *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
- **properties** – the properties of the model

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

Transform the sqlmodel to view model

Parameters

- **namespace** – Namespace of the model
- **tablename** – Name od 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 bloks

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(*registry*, *namespace*, *base*, *properties*)
Find the sqlalchemy hybrid methods in the base to save the namespace and the method in the 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

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*, *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
- **properties** – the properties of the model

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 known 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 od 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

class Declarations.**RelationShip**(**args*, ***kwargs*)

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*)

Add the InstrumentedList class to replace List class as result of the query

Parameters **registry** – current registry

check_existing_remote_model(*registry*)

Check if the remote model exists

The information of the existance come from the first step of assembling

Exception FieldException if the model doesn't exist

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

find_primary_key(*properties*)

Return the primary key come from the first step property

Parameters **properties** – first step properties for the model

Return type column name of the primary key

Exception FieldException

format_backref(*registry*, *namespace*, *properties*)

Create the real backref, with the backref string and the backref properties

Parameters

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

get_registry_name()

Return the registry name of the remote model

Return type str of the registry name

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 relation ship instance

get_tablename(*registry*, *model=None*)
Return the table name of the remote model

Return type str of the table name

must_be_declared_as_attr()
Return True, because it is a relationship

classmethod Declarations.**add_declaration_type**(*cls_=None*, *isAnEntry=False*, *assemble=None*, *initialize=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)

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_

7.3 anyblok._argsparse module

class anyblok._argsparse.**ArgsParseManager**
ArgsParse 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 ArgsParser after loading

classmethod add(*group*, *part='AnyBlok'*, *label=None*, *function_=None*)

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

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

- or by calling the `add` method as a decorator:

```
@ArgsParseManager.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:

```
@ArgsParseManager.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

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 ArgsParseManager. And all the applications have free access to these options:

```
from anyblok._argsparse import ArgsParseManager

database = ArgsParseManager.get('dbname')
```

..warning:

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

Parameters

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

classmethod get_url (*dbname=None*)
Return an sqlalchemy URL for database

Get the options of the database, the only option which can be overloaded is the name of the database:

```
url = ArgsParseManager.get_url(dbname='Mydb')
```

Parameters **dbname** – Name of the database

Return type SqlAlchemy URL

Exception ArgsParseManagerException

classmethod load (*description='AnyBlok :'*, *argsparse_groups=None*, *parts_to_load=None*, *useseparator=False*)
Load the argparse definition and parse them

Parameters

- **description** – description of argsparse
- **argsparse_groups** – list argsparse groupe to load
- **parts_to_load** – group of blok to load
- **useseparator** – boolean(default False)

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

ArgsParseManager.add('create-db', function_=foo)
ArgsParseManager.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='AnyBlok'*)
Remove an existing label

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

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

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

Parameters

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

7.4 anyblok._imp module

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

7.5 anyblok._logging module

class anyblok._logging.**Formatter**(*fmt=None*, *datefmt=None*, *style='%'*)
Bases: logging.Formatter

Define the format for console logging

format(*record*)
Add color to the message

Parameters **record** – logging record instance

Return type logging record formatted

anyblok._logging.**init_logger**(*level='info'*, *mode='console'*, *filename=None*, *socket=None*, *facility=8*)
Init the logger output

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

Example:

```
from anyblok.log import init_logger
init_logger(level='debug')
```

A logger can log to:

•console (default):

```
init_logger(mode='console')
```

•file:

```
init_logger(mode='file', filename='my.file.log')
```

•socket:

```
init_logger(mode='socket', socket=('localhost', 1000))
```

•syslog:

Example:

```
# By socket
init_logger(mode='syslog', socket('localhost', 514))
# By UNIX socket
init_logger(mode='syslog', socket='/dev/log')
```

the syslog mode define logger facility:

–LOG_AUTH

–LOG_AUTHPRIV

–LOG_CRON

–LOG_DAEMON

–LOG_FTP

–LOG_KERN

–LOG_LPR

–LOG_MAIL

–LOG_NEWS

–LOG_SYSLOG

–LOG_USER (default)

–LOG_UUCP

–LOG_LOCAL0

–LOG_LOCAL1

–LOG_LOCAL2

–LOG_LOCAL3

–LOG_LOCAL4

–LOG_LOCAL5

–LOG_LOCAL6

–LOG_LOCAL7

example:

```
init_logger(mode='syslog', socket='/dev/log',
            facility=syslog.LOG_SYSLOG)
```

Parameters

- **level** – level defined by anyblok
- **mode** – Output mode
- **filename** – Output file
- **socket** – Socket or UnixSocket
- **facility** –

Exception Exception

anyblok._logging.**log**(*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:

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

Parameters

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

7.6 anyblok.environment module

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

7.7 anyblok.blok module

class anyblok.blok.**BlokManager**
Manage the bloks for one process

A blok has a *setuptools* entrypoint, this entry point is defined by the `bloks_groups` 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('AnyBlok')
```

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 **has**(*blok*)
Return True if the blok is loaded

Parameters **blok** – blok name

Return type bool

classmethod **list**()
Return the ordered bloks

Return type list of blok name ordered by loading

classmethod **load**(**bloks_groups*)
Load all the bloks and import them

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

Exception BlokManagerException

classmethod **reload**()
Reload the entry points

Empty the bloks dict and use the bloks_groups 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

•conditionnal: if all the bloks of this list are installed then install this blok

7.8 anyblok.registry module

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 **get**(*dbname*)
Return an existing Registry

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

Parameters **dbname** – 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 bloks loaded must be initialized because the registry will be created with this information

Parameters **blokname** – name of the blok

classmethod reload(*blok*)
Reload the blok

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

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

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

class anyblok.registry.**Registry**(*dbname*)
Define one registry

A registry is linked to a database, and stores the definition of the installed Bloks, 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

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

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

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

precommit_hook(*registryname*, *method*, *put_at_the_if_exist*)
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_if_exist** – if true and hook allready exist then the hook are moved at the end

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

7.9 anyblok.migration module

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

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**(*column*)
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(*remote_field*, ***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(Sqlachemy 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 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

foreign_key()
Get a foreign key

Return type MigrationConstraintForeignKey instance

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(*name*, *condition*)
Add the constraint

Parameters

- **name** – name of the constraint
- **condition** – constraint to apply

Return type MigrationConstraintCheck instance

drop()
Drop the constraint

class anyblok.migration.**MigrationConstraintUnique**(*table*, **columns*, ***kwargs*)
Used for the Unique constraint

Add a new constraint:

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

Get and drop the constraint:

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

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*)
Atler 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

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(**columns*, ***kwargs*)
Get unique

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

Return type MigrationConstraintUnique instance

class anyblok.migration.**Migration**(*session*, *metadata*)
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 automaticly

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

7.10 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

7.11 anyblok.databases module

Management of the database

```
adapter = getUtility(ISqlAlchemyDataBase, drivername)
adapter.createdb(dbname)
logger.info(adapter.listdb())
adapter.dropdb(dbname)
```

7.11.1 anyblok.databases.postgres module

class anyblok.databases.postgres.**SqlAlchemyPostgres**
Postgres adapter for database management

cnx()
Context manager to get a connection to database

createdb(*dbname*)
Create a database

Parameters **dbname** – database name to create

dropdb(*dbname*)
Drop a database

Parameters **dbname** – database name to drop

listdb()
list database

Return type list of database name

7.12 anyblok.scripts module

anyblok.scripts.**createdb**(*description*, *argsparse_groups*, *parts_to_load*)
Create a database and install blok from config

Parameters

- **description** – description of argsparse
- **argsparse_groups** – list argsparse groupe to load
- **parts_to_load** – group of blok to load

anyblok.scripts.**updatedb**(*description*, *version*, *argsparse_groups*, *parts_to_load*)
Update an existing database

Parameters

- **description** – description of argsparse
- **version** – version of script for argparse
- **argsparse_groups** – list argsparse groupe to load
- **parts_to_load** – group of blok to load

anyblok.scripts.**interpreter**(*description*, *version*, *argsparse_groups*, *parts_to_load*)
Execute a script or open an interpreter

Parameters

- **description** – description of argsparse
- **version** – version of script for argparse
- **argsparse_groups** – list argsparse groupe to load
- **parts_to_load** – group of blok to load

anyblok.scripts.**sqlschema**(*description*, *version*, *argsparse_groups*, *parts_to_load*)
Create a Table model schema of the registry

Parameters

- **description** – description of argsparse
- **version** – version of script for argparse
- **argsparse_groups** – list argsparse groupe to load
- **parts_to_load** – group of blok to load

anyblok.scripts.**modelschema**(*description*, *version*, *argsparse_groups*, *parts_to_load*)
Create a UML model schema of the registry

Parameters

- **description** – description of argsparse
- **version** – version of script for argparse
- **argsparse_groups** – list argsparse groupe to load
- **parts_to_load** – group of blok to load

Contents

- *Helper for unittest*
  - *TestCase*
  - *DBTestCase*
  - *BlokTestCase*

CHAPTER 8

Helper for unittest

For unittest, classes are available to offer some fonctionnalities

8.1 TestCase

```
from anyblok.tests.testcase import TestCase
```

class anyblok.tests.testcase.**TestCase**(*methodName='runTest'*)

Bases: unittest.case.TestCase

Unittest class add helper for unit test in anyblok

classmethod **createdb**(*keep_existing=False*)

Create a database in fonction of variable of environment

```
        cls.init_argsparse_manager()
        cls.createdb()
```

Parameters **keep_existing** – If false drop the previous db before create it

classmethod **dropdb**()

Drop a database in fonction of variable of environment

```
        cls.init_argsparse_manager()
        cls.dropdb()
```

getRegistry()

Return the registry for the database in argsparse i

```
        registry = self.getRegistry()
```

Return type registry instance

classmethod **init_argsparse_manager**(*prefix=None*, ***env*)

Initialise the argsparse 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

8.2 DBTestCase

Warning: this testcase destroys the test database for each unittest

class anyblok.tests.testcase.**DBTestCase**(*methodName='runTest'*)
Bases: anyblok.tests.testcase.TestCase

Test case for all the Field, Column, RelationShip

```
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)
```

Warning: The database are create and drop for each unit test

current_blok = 'anyblok-core'
In the blok to add the new model

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

parts_to_load = ['AnyBlok']
blok group to load

setUp()
Create a database and load the blok manager

classmethod setUpClass()
Intialialise the argsparse manager

tearDown()
Clear the registry, unload the blok manager and drop the database

upgrade(*registry*, ***kwargs*)
Upgrade the registry:

```
class MyTest(DBTestCase):

    def test_mytest(self):
        registry = self.init_registry(...)
        self.upgrade(registry, install=('MyBlok',))
```

Parameters

- **registry** – registry to upgrade
- **install** – list the blok to install
- **update** – list the blok to update
- **uninstall** – list the blok to uninstall

8.3 BlokTestCase

class anyblok.tests.testcase.**BlokTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Use to test bloks without have to create new database for each test

```
from anyblok.tests.testcase import BlokTestCase


class MyBlokTest(BlokTestCase):

    def test_1(self):
        # access of the registry by ``self.registry``
        ...
```

classmethod setUpClass()
Intialialise the argsparse manager

Deactivate the commit method of the registry

tearDown()
Roll back the session

classmethod tearDownClass()

Contents

- *Bloks*
  - *anyblok-core blok*

CHAPTER 9

Bloks

9.1 anyblok-core blok

Contents

- *CHANGELOG*
  - *Future*
  - *0.2.3*
  - *0.2.2*
  - *0.2.0*
  - *0.1.3*
  - *0.1.2*
  - *0.1.1*
  - *0.1.0*

CHAPTER 10

CHANGELOG

10.1 Future

- [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)

10.2 0.2.3

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

10.3 0.2.2

- **[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

10.4 0.2.0

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

10.5 0.1.3

- [FIX] setup long description, good for pypi but not for easy_install

10.6 0.1.2

- [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

10.7 0.1.1

- [FIX] version, documentation, setup

10.8 0.1.0

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*
  - *Next step for the 0.2*
  - *To implement*
  - *Library to include*
  - *Functionnality which need a sprint*

CHAPTER 11

ROADMAP

11.1 Next step for the 0.2

- Access Rules / Roles
- Add logo and slogan
- Update doc

11.2 To implement

- Add RelationShip model in anyblok-core and refactor the get column http://docs.sqlalchemy.org/en/latest/faq.html#how-do-i-get-a-list-of-all-columns-relationships-mapped-attributes-etc-given-a-mapped-class
- Put postgres database in his own distribution with the good import
- Need improve alembic

11.3 Library to include

- Addons for sqlalchemy : http://sqlalchemy-utils.readthedocs.org/en/latest/installation.html
- 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

11.4 Functionnality which need a sprint

- Back Task
- Cron
- Tasks Management

- Event by messaging bus
- Import / Export
- Internalization
- Ancestor left / right
- Access Rules / Roles

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*

CHAPTER 12

Mozilla Public License Version 2.0

12.1 1. Definitions

12.1.1 1.1. “Contributor”

Means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.

12.1.2 1.2. “Contributor Version”

Means the combination of the Contributions of others (if any) used by a Contributor and that particular Contributor’s Contribution.

12.1.3 1.3. “Contribution”

Means Covered Software of a particular Contributor.

12.1.4 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.

12.1.5 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.

12.1.6 1.6. “Executable Form”

Means any form of the work other than Source Code Form.

12.1.7 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.

12.1.8 1.8. “License”

Means this document.

12.1.9 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.

12.1.10 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.

12.1.11 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.

12.1.12 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.

12.1.13 1.13. “Source Code Form”

Means the form of the work preferred for making modifications.

12.1.14 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.

12.2 2. License Grants and Conditions

12.2.1 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.

12.2.2 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.

12.2.3 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).

12.2.4 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).

12.2.5 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.

12.2.6 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.

12.2.7 2.7. Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

12.3 3. Responsibilities

12.3.1 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.

12.3.2 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.

12.3.3 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).

12.3.4 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.

12.3.5 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.

12.4 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.

12.5 5. Termination

12.5.1 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.

12.5.2 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.

12.5.3 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.

12.6 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.

12.7 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.

12.8 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.

12.9 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.

12.10 10. Versions of the License

12.10.1 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.

12.10.2 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.

12.10.3 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).

12.10.4 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.

12.11 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.

12.12 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 13

Indices and tables

- genindex
- modindex
- search

a

anyblok, 47
anyblok._argsparse, 54
anyblok._graphviz, 71
anyblok._imp, 57
anyblok.blok, 60
anyblok.databases.postgres, 74
anyblok.declarations, 47
anyblok.environment, 59
anyblok.migration, 66
anyblok.registry, 61
anyblok.scripts, 74
anyblok.tests.testcase, 77

A

anyblok (module), 47
anyblok._argsparse (module), 54
anyblok._graphviz (module), 71
anyblok._imp (module), 57
anyblok.blok (module), 60
anyblok.databases.postgres (module), 74
anyblok.declarations (module), 47
anyblok.environment (module), 59
anyblok.migration (module), 66
anyblok.registry (module), 61
anyblok.scripts (module), 74
anyblok.tests.testcase (module), 77