almir Documentation

Release 0.1.6

Domen Kožar

March 26, 2013

CONTENTS

Author Domen Kožar <domen@dev.si>

Source code github.com project

Bug tracker github.com issues

Generated March 26, 2013

License GPL v3

Version 0.1.6

Features

- supports bacula-director version 5.x.x
- supports sqlite, postgresql, mysql databases
- complete read-only interface for bacula-director
- interactive web console frontend to bconsole
- export data to excel, pdf and clipboard
- supports python2.6 and python 2.7
- 100% test coverage

Overview

Almir is a bacula web interface for administrators written in Python. It is designed with simplicity in mind, although backup management is never simple.
It is named after Almir Karič, a fellow developer from **Slovenia**, whose dream was to live in San Francisco. Two years after he moved, he died in a car crash. This is his gift for eternity.
Almir is open source software licensed under **GPL v3**, started by Domen Kožar in 2011.

CHAPTER

ONE

USER GUIDE

1.1 Demo

http://almir-demo.domenkozar.com/

1.2 Design goals

- each release is pinned to bacula-director specific major version
- simple is better than complicated
- inline documentation
- convention over configuration
- can act as view interface only (optional configuration functionality)
- plugin into existing bacula instance
- total control of bacula through existing bacula api

1.3 Installation

Almir will install everything inside one directory, which must be empty. Application is meant to be self-contained, meaning no additional administrator is needed besides upgrading to a newer version. Almir should be always installed on a system together with bacula-director.

1.3.1 Prerequisities

- Bacula 5.x.x is installed and bacula-director is running
- installed python2.6 or python2.7 (compiled with sqlite support)
- using postgresql: make sure *postgresql.conf* includes a line *client_encoding = utf8*

1.3.2 Installer (interactive)

Install prerequesities (Debian based):

```
$ sudo apt-get install git bacula-console python-distribute gcc python-dev wget
```

Note: Installer will ask you few questions about SQL database and configuration for bconsole.

Install almir (recommended: under same user as bacula):

```
$ cd /some/empty/directory/to/install/almir/
$ sh -xec "$(wget -O - https://raw.github.com/iElectric/almir/master/install_production.sh)"
```

You can continue with configuring *Configuring Nginx as a frontend*.

1.3.3 Manual (not recommended)

For security and *unix freaks, here is a step by step description what interactive install does behind the scene. Taking manual steps to install will ensure you are missing lovely time with your beloved one this weekend and replacing that with mild headache (specially if you are not familiar with python deployment quirks).

Interactive install also handles upgrades transparently. Almir is developed with agile workflow with small incremental versions of new changes. You will have to dig in yourself how to upgrade environment upon a new release. Still stubborn? Let's go!

- Download latests release from github (as tarball or git clone) from *latests* branch (install_production.sh takes care of that otherwise)
- unpack into empty directory
- Make sure you use python2.7 or python2.6, since other versions are not supported
- Populate *production.ini* file for daemon configuration, taken from *buildout.d/templates/production.ini.in* (*almir/scripts/configure_deploy.py* takes care of that interactively, then runs buildout to output configuration file from the template)
- Install all python dependencies from *setup.py* file, preferably within virtualenv (buildout takes care of that and pins them down to known workable set of versions, found in *buildout.d/versions.cfg*)
- Once you have installed dependencies (with *python setup.py install*) inside virtualenv or system python (REALLY not recommended), you should have *pserve* binary installed in bin/ directory
- run it like so: *bin/pserve production.ini*
- make sure daemon runs at reboot, configure log rotation

Happy? Let's see until first upgrade.

1.3.4 Configuring Nginx as a frontend

It is wise to use frontend HTTP server and proxy HTTP requests to python web server. Following is an example for nginx, you could also use papache2 or lighthttpd.

You would normally put this in /etc/nginx/sites-enabled/almir.mywebsite.com.conf

```
server {
    listen          80;
    server_name     almir.mywebsite.com;

    location / {
            proxy_pass http://localhost:2500;

            proxy_set_header  X-Real-IP  $remote_addr;
            proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header  Host $http_host;
            proxy_redirect off;

            # optional authentication - recommended
            auth_basic "Restricted";
            # how to correctly write htpasswd: http://wiki.nginx.org/HttpAuthBasicModule#auth_basic_u
            auth_basic_user_file /some/directory/to/install/almir/.htpasswd;

    }
}
```

Then run:

```
$ /etc/init.d/nginx reload
```

Now try to access http://almir.mywebsite.com/ (if you have an error, follow instructions at *Reporting bugs*)

1.3.5 Configuring Apache2 as a frontend

```
<VirtualHost *:80>

ServerName almir.mydomain.com
DocumentRoot "/var/www/almir.mydomain.com"

ProxyPreserveHost On

<Location />
ProxyPass  http://almir.mydomain.com:2500/
ProxyPassReverse http://almir.mydomain.com:2500/
</Location>

ErrorLog /var/log/httpd/almir.mydomain.com-error.log
CustomLog /var/log/httpd/almir.mydomain.com-access.log combined

</VirtualHost>
```

Do not forget to restrict access to almir, either by IP or by username/password.

1.4 Upgrading to a newer release

Run:

```
$ cd almir_install_directory
$ git pull
$ python bootstrap.py
$ bin/buildout
```

```
$ bin/supervisorctl shutdown
$ bin/supervisord
```

You can also use that in crontab to auto upgrade on new releases, if you are crazy enough. You would probably extra check if upgrade is needed, something like running following and checking for any output:

```
$ git log latests..origin/latests
```

1.5 Reporting bugs

Check if an issue already exists at https://github.com/iElectric/almir/issues, otherwise add new one with following information:

- bacula-director version, operating system and browser version
- include screenshot if it provides any useful information
- pastebin (http://paste2.org) output of $ cat ALMIR_ROOT/var/logs/almir-stderr*
- pastebin ALMIR_ROOT/buildout.cfg, but be careful to *remove any sensitive data*

1.6 Filesystem structure

TODO ;)

CHAPTER

TWO

DEVELOPER GUIDE

2.1 Setup developer environment

- sudo aptitude install git gcc python-dev bacula-console
- git clone https://github.com/iElectric/almir.git almir
- cd almir
- cp buildout.d/templates/buildout.cfg.in buildout.cfg
- vim buildout.cfg # configure variables
- python bootstrap.py
- bin/buildout
- bin/pserve –reload development.ini

2.2 Running Python tests

Easy as:

```
$ bin/nosetests
```

By default it will run against sqlite fixture, you can also tell nosetests to use mysql fixture (you need to import sql manually for now):

```
$ DATABASE="mysql" bin/nosetests
```

Or just specify sqlalchemy engine:

```
$ ENGINE="sqlite:////var/lib/bacula/bacula.db" bin/nosetests
```

2.3 Running Javascript tests

Install and configure phantomjs (webkit headless testing):

```
$ sudo apt-get install libqtwebkit-dev
$ git clone git://github.com/ariya/phantomjs.git && cd phantomjs
$ qmake-qt4 && make
$ sudo cp bin/phantomjs /usr/local/bin/
```

Run tests:

```
$ cd ../almir
$ ./.travis_qunit_tests.sh
```

2.4 Coding conventions

- PEP8 except for 80 char length rule
- add changelog, test and documentation with code in commits
- same applies to javascript
- jslinted javascript

2.5 Releasing `almir`

```
$ bin/fullrelease
$ git checkout latests
$ git merge master
$ git push
# update http://readthedocs.org/dashboard/almir/versions/
```

CHAPTER

THREE

CHANGELOG

3.1 0.1.6 (2013-03-27)

- [bug] Add .ini to MANIFEST.in [Domen Kožar]

3.2 0.1.5 (2013-03-27)

- [feature] Refactor the package a bit, so it's easier to package it for Linux distributions [Domen Kožar]
- [bug] Update MANIFEST.in so all files are included in release [Domen Kožar]
- [bug] Add new bootstrap.py and pin down zc.buildout version to avoid upgrading zc.buildout to 2.0 [Domen Kožar]
- [feature] Add apache2 configuration example [Iban]

3.3 0.1.4 (2013/03/23)

- brownbag release

3.4 0.1.3 (2012/08/27)

- [bug] upgraded doctutils as it was failing buildout [Domen Kožar]
- removed some dependencies on production, upgraded zc.buildout to 1.6.3 for faster installation [Domen Kožar]
- determine version from distribution metadata [Domen Kožar]

3.5 0.1.2 (2012/05/31)

- [bug] interactive installer would swallow error message when SQL connection string was not formatted correctly
- [bug]: #7: don't word wrap size columns
- [feature] add manual install steps
- [bug] #4: client detail page did not render if client had no successful backup

- [bug] #5: correctly parse scheduled jobs (choked on Admin job)
- [feature] use python2.7 or python2.6, otherwise abort installation

3.6 0.1.1 (2012/04/18)

- [bug] fix support for postgresql 9.1 [Domen Kožar]
- [feature] add reboot crontab for almir daemon [Domen Kožar]
- [bug] MySQL database size calculation was wrong, sometimes crashing the dashboard [Domen Kožar]
- [bug] console command list was not ordered and search box was not shown [Domen Kožar]
- [bug] bconsole did not accept non-asci input [Domen Kožar]

3.7 0.1 (2012/04/06)

- Initial version [Domen Kožar]

CHAPTER

FOUR

SOURCE DOCUMENTATION

4.1 almir – Main package

4.1.1 almir.forms – HTML forms definitions

class almir.forms.**JobSchema**(**arg*, ***kw*)
Bases: colander.Schema

class almir.forms.**LogSchema**(**arg*, ***kw*)
Bases: colander.Schema

class almir.forms.**MediaSchema**(**arg*, ***kw*)
Bases: colander.Schema

almir.forms.**deferred_widget_factory**(*values_name*)

4.1.2 almir.meta – Configuration for models

class almir.meta.**ModelMixin**
Bases: object

static format_byte_size(*size*)

classmethod get_list(***kw*)

classmethod get_one(*id_=None*, *query=None*)

query = None

static render_distance_of_time_in_words(*dt_from*, *dt_to=None*)

almir.meta.**get_database_size**(*engine*)
Returns human formatted SQL database size.

Example: 3.04 GB

almir.meta.**initialize_sql**(*settings*)

4.1.3 almir.models – SQLAlchemy models

Models generated from bacula-dir-postgresql 5.0.2

class almir.models.**Client**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+———-+———————————————————— clientid | integer | not null default nextval('client_clientid_seq'::regclass) name | text | not null uname | text | not null autoprune | smallint | default 0 fileretention | bigint | default 0 jobretention | bigint | default 0

Indexes: "client_pkey" PRIMARY KEY, btree (clientid) "client_name_idx" UNIQUE, btree (name)

classmethod **get_list**(***kw*)

render_autoprune(*request*)

render_fileretention(*request*)

render_jobretention(*request*)

render_name(*request*)

url(*request*)

class almir.models.**File**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+———+———————————————————- fileid | bigint | not null default nextval('file_fileid_seq'::regclass) fileindex | integer | not null default 0 jobid | integer | not null pathid | integer | not null filenameid | integer | not null markid | integer | not null default 0 lstat | text | not null md5 | text | not null

Indexes: "file_pkey" PRIMARY KEY, btree (fileid) "file_jobid_idx" btree (jobid) "file_jpfid_idx" btree (jobid, pathid, filenameid)

filename

get_stat_data()

path

render_filename(*request*)

render_gid(*request*)

render_mode(*request*)

render_size(*request*)

render_uid(*request*)

class almir.models.**FileSet**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+—————————+———————————————————- filesetid | integer | not null default nextval('fileset_filesetid_seq'::regclass) fileset | text | not null md5 | text | not null createtime | timestamp without time zone | not null

Indexes: "fileset_pkey" PRIMARY KEY, btree (filesetid) "fileset_name_idx" btree (fileset)

createtime = Column('createtime', BaculaDateTime(), table=<FileSet>)

class almir.models.**Filename**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————+———+———————————————————— filenameid | integer | not null default nextval('filename_filenameid_seq'::regclass) name | text | not null

Indexes: "filename_pkey" PRIMARY KEY, btree (filenameid) "filename_name_idx" btree (name)

class `almir.models.`**`Job`**(***kwargs*)

Bases: `almir.meta.ModelMixin`, `sqlalchemy.ext.declarative.api.Base`

Column | Type | Modifiers

————————+—————————+———————————————————— jobid | integer | not null default nextval('job_jobid_seq'::regclass) job | text | not null name | text | not null type | character(1) | not null level | character(1) | not null clientid | integer | default 0 jobstatus | character(1) | not null schedtime | timestamp without time zone | starttime | timestamp without time zone | endtime | timestamp without time zone | realendtime | timestamp without time zone | jobtdate | bigint | default 0 volsessionid | integer | default 0 volsessiontime | integer | default 0 jobfiles | integer | default 0 jobbytes | bigint | default 0 readbytes | bigint | default 0 joberrors | integer | default 0 jobmissingfiles | integer | default 0 poolid | integer | default 0 filesetid | integer | default 0 priorjobid | integer | default 0 purgedfiles | smallint | default 0 hasbase | smallint | default 0 hascache | smallint | default 0 reviewed | smallint | default 0 comment | text |

Indexes: "job_pkey" PRIMARY KEY, btree (jobid) "job_name_idx" btree (name)

`client`

`endtime` = Column('endtime', BaculaDateTime(), table=<Job>)

`files`

classmethod **`get_last`**()

classmethod **`get_list`**(***kw*)

classmethod **`get_one`**(*id_*)

classmethod **`get_running`**()

classmethod **`get_upcoming`**()

`jobmedias`

`logs`

`pool`

`realendtime` = Column('realendtime', BaculaDateTime(), table=<Job>)

`render_client_name`(*request*)

`render_duration`(*request*)

`render_jobbytes`(*request*)

`render_joberrors`(*request*)

`render_jobfiles`(*request*)

`render_level`(*request*)

`render_name`(*request*)

`render_pool_name`(*request*)

`render_starttime`(*request*)

render_status(*request*)

render_status_color(*request*)
Color of the job depending on status

render_type(*request*)

render_volume_name(*request*)

schedtime = Column('schedtime', BaculaDateTime(), table=<Job>)

starttime = Column('starttime', BaculaDateTime(), table=<Job>)

status

url(*request*)

class almir.models.**JobMedia**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+———+———————————————————— jobmediaid | integer | not null default nextval('jobmedia_jobmediaid_seq'::regclass) jobid | integer | not null mediaid | integer | not null firstindex | integer | default 0 lastindex | integer | default 0 startfile | integer | default 0 endfile | integer | default 0 startblock | bigint | default 0 endblock | bigint | default 0 volindex | integer | default 0

Indexes: "jobmedia_pkey" PRIMARY KEY, btree (jobmediaid) "job_media_job_id_media_id_idx" btree (jobid, mediaid)

medias

class almir.models.**Log**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+—————————————+———————————————————— logid | integer | not null default nextval('log_logid_seq'::regclass) jobid | integer | not null time | timestamp without time zone | logtext | text | not null

Indexes: "log_pkey" PRIMARY KEY, btree (logid) "log_name_idx" btree (jobid)

classmethod get_list(***kw*)

render_jobid(*request*)

render_logtext(*request*)

render_time(*request*)

time = Column('time', BaculaDateTime(), table=<Log>)

class almir.models.**Media**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————————+—————————————+———————————————————— actiononpurge | smallint | default 0 comment | text | deviceid | integer | default 0 enabled | smallint | default 1 endblock | bigint | default 0 endfile | integer | default 0 firstwritten | timestamp without time zone | inchanger | smallint | default 0 initialwrite | timestamp without time zone | labeldate | timestamp without time zone | labeltype | integer | default 0 lastwritten | timestamp without time zone | locationid | integer | default 0 maxvolbytes

| bigint | default 0 maxvolfiles | integer | default 0 maxvoljobs | integer | default 0 mediaaddressing | smallint | default 0 mediaid | integer | not null default nextval('media_mediaid_seq'::regclass) mediatypeid | integer | default 0 mediatype | text | not null poolid | integer | default 0 recyclecount | integer | default 0 recyclepoolid | integer | default 0 recycle | smallint | default 0 scratchpoolid | integer | default 0 slot | integer | default 0 storageid | integer | default 0 volblocks | integer | default 0 volbytes | bigint | default 0 volcapacitybytes | bigint | default 0 volerrors | integer | default 0 volfiles | integer | default 0 voljobs | integer | default 0 volmounts | integer | default 0 volparts | integer | default 0 volreadtime | bigint | default 0 volretention | bigint | default 0 volstatus | text | not null volumename | text | not null voluseduration | bigint | default 0 volwrites | integer | default 0 volwritetime | bigint | default 0

Indexes: "media_pkey" PRIMARY KEY, btree (mediaid) "media_volumename_id" UNIQUE, btree (volumename)

Check constraints: "media_volstatus_check" CHECK (volstatus = ANY (ARRAY['Full'::text, 'Archive'::text, 'Append'::text, 'Recycle'::text, 'Purged'::text, 'Read-Only'::text, 'Disabled'::text, 'Error'::text, 'Busy'::text, 'Used'::text, 'Cleaning'::text, 'Scratch'::text]))

firstwritten = Column('firstwritten', BaculaDateTime(), table=<Media>)

classmethod **get_list**(***kw*)

classmethod **get_one**(*id_*)

initialwrite = Column('initialwrite', BaculaDateTime(), table=<Media>)

jobs

labeldate = Column('labeldate', BaculaDateTime(), table=<Media>)

lastwritten = Column('lastwritten', BaculaDateTime(), table=<Media>)

pool

render_enabled(*request*)

render_expires(*request*)

render_maxvolbytes(*request*)

render_pool_name(*request*)

render_recycle(*request*)

render_storage_name(*request*)

render_volbytes(*request*)

render_volcapacitybytes(*request*)

render_volretention(*request*)

render_volstatus(*request*)

render_volumename(*request*)

storage

url(*request*)

class almir.models.**Path**(***kwargs*)

Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

———+———+———————————————- pathid | integer | not null default nextval('path_pathid_seq'::regclass) path | text | not null

Indexes: “path_pkey” PRIMARY KEY, btree (pathid) “path_name_idx” btree (path)

class almir.models.**Pool**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+———+————————————————— acceptanyvolume | smallint | default 0 actiononpurge | smallint | default 0 autoprune | smallint | default 0 enabled | smallint | default 1 labelformat | text | not null labeltype | integer | default 0 maxvolbytes | bigint | default 0 maxvolfiles | integer | default 0 maxvoljobs | integer | default 0 maxvols | integer | default 0 migrationhighbytes | bigint | default 0 migrationlowbytes | bigint | default 0 migrationtime | bigint | default 0 name | text | not null nextpoolid | integer | default 0 numvols | integer | default 0 poolid | integer | not null default nextval(‘pool_poolid_seq’::regclass) pooltype | text | recyclepoolid | integer | default 0 recycle | smallint | default 0 scratchpoolid | integer | default 0 usecatalog | smallint | default 0 useonce | smallint | default 0 volretention | bigint | default 0 voluseduration | bigint | default 0

Indexes: “pool_pkey” PRIMARY KEY, btree (poolid) “pool_name_idx” btree (name)

Check constraints: “pool_pooltype_check” CHECK (pooltype = ANY (ARRAY[’Backup’::text, ‘Copy’::text, ‘Cloned’::text, ‘Archive’::text, ‘Migration’::text, ‘Scratch’::text]))

classmethod **get_one**(*id_*)

classmethod **get_values**()

render_acceptanyvolume(*request*)

render_autoprune(*request*)

render_enabled(*request*)

render_name(*request*)

render_recycle(*request*)

render_usecatalog(*request*)

render_useonce(*request*)

url(*request*)

class almir.models.**Status**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+————————+———— jobstatus | character(1) | not null jobstatuslong | text | severity | integer |

Indexes: “status_pkey” PRIMARY KEY, btree (jobstatus)

classmethod **get_values**()

class almir.models.**Storage**(***kwargs*)
Bases: almir.meta.ModelMixin, sqlalchemy.ext.declarative.api.Base

Column | Type | Modifiers

————————+———+————————————————— storageid | integer | not null default nextval(‘storage_storageid_seq’::regclass) name | text | not null autochanger | integer | default 0

Indexes: “storage_pkey” PRIMARY KEY, btree (storageid)

classmethod get_list(***kw*)

classmethod get_one(*id_*)

classmethod get_values()

render_autochanger(*request*)

render_name(*request*)

url(*request*)

4.1.4 almir.views – Pyramid views

class almir.views.**ClientView**(*request*)
Bases: almir.views.MixinView

detail()

model
alias of Client

class almir.views.**JobView**(*request*)
Bases: almir.views.MixinView

detail()

list()

model
alias of Job

schema
alias of JobSchema

class almir.views.**LogView**(*request*)
Bases: almir.views.MixinView

model
alias of Log

schema
alias of LogSchema

class almir.views.**MixinView**(*request*)
Bases: object

detail()

get_form()
Deals everything regarding forms for a request.

list()

model = None

schema = None

class almir.views.**PoolView**(*request*)
Bases: almir.views.MixinView

model
alias of Pool

class almir.views.**StorageView**(*request*)
Bases: almir.views.MixinView

model
alias of Storage

class almir.views.**VolumeView**(*request*)
Bases: almir.views.MixinView

list()

model
alias of Media

schema
alias of MediaSchema

almir.views.**about**(*request*)

almir.views.**ajax_console_input**(*request*)

almir.views.**console**(*request*)

almir.views.**dashboard**(*request*)

almir.views.**datatables**(*request*)
Implements server side interface for datatables.

For field reference look at http://www.datatables.net/usage/server-side

almir.views.**httpexception**(*context*, *request*)

4.1.5 almir.lib – Non MVC code

almir.lib.bacula_base64 – Bacula custom base64 implmentation

Taken from https://github.com/ZungBang/baculafs/blob/master/baculafs/Base64.py under GPLv3

almir.lib.bacula_base64.**decode_base64**(*base64*)
Bacula specific implementation of a base64 decoder

almir.lib.bconsole – Python interface to bconsole

class almir.lib.bconsole.**BConsole**(*bconsole_command='bconsole -n -c %s'*, *config_file=None*)
Bases: object

Interface to bconsole binary

classmethod from_temp_config(**args*, ***kwds*)
Constructs BConsole object with help of passing temporary file for the session.

get_jobs_settings()

get_upcoming_jobs(*days=1*)

get_version()

is_running()

make_backup(*job*, *level=None*, *storage=None*, *fileset=None*, *client=None*, *priority=None*, *pool=None*, *when=None*)

send_command_by_polling(*command*, *process=None*)

start_process()

exception almir.lib.bconsole.**BConsoleError**
Bases: exceptions.Exception

exception almir.lib.bconsole.**DirectorNotRunning**
Bases: almir.lib.bconsole.BConsoleError

almir.lib.console_commands – Parsed list of bconsole commands

almir.lib.sqlalchemy_custom_types

class almir.lib.sqlalchemy_custom_types.**BaculaDateTime**(**args*, ***kwargs*)
Bases: sqlalchemy.types.TypeDecorator

Changes sqlite DateTime to parse 0 values as no value. Also converts to right timezone

impl
alias of DateTime

process_result_value(*value*, *dialect=None*)

result_processor(*dialect*, *coltype*)

almir.lib.sqlalchemy_declarative_reflection

class almir.lib.sqlalchemy_declarative_reflection.**DeclarativeReflectedBase**
Bases: object

classmethod prepare(*engine*)
Reflect all the tables and map !

almir.lib.sqlalchemy_lowercase_inspector

class almir.lib.sqlalchemy_lowercase_inspector.**LowerCaseInspector**(*bind*)
Bases: sqlalchemy.engine.reflection.Inspector

Implements reflection inspector that reflects everything lowercase

get_columns(**a*, ***kw*)

get_foreign_keys(**a*, ***kw*)

get_indexes(**a*, ***kw*)

get_pk_constraint(**a*, ***kw*)

almir.lib.utils - General utilities

almir.lib.utils.**convert_timezone**(*datetime*)
Converts datetime to timezone aware datetime.

Retrieving timezone:

- get *timezone* from .ini settings
- default to system timezone

almir.lib.utils.**get_jinja_macro**(*macro*)
Return actual function from a jinja2 template

almir.lib.utils.**nl2br**(*text*)

almir.lib.utils.**render_rst_section**(*filename*)
Finds filename in documentation directory and renders it to html.

almir.lib.utils.**timedelta_to_seconds**(*td*)

almir.lib.utils.**yesno**(*text*)

4.1.6 almir.scripts – Runnable scripts package

almir.scripts.configure_deploy – Ask few questions and configure almir

Dirty script to output buildout.cfg, but it does the job.

almir.scripts.configure_deploy.**ask_question**(*question*, *default=None*, *validator=None*, *func=<built-in function raw_input>*)

almir.scripts.configure_deploy.**main**()
Entry point of this script

almir.scripts.configure_deploy.**validate_engine**(*v*)

almir.scripts.configure_deploy.**validate_int**(*v*)

almir.scripts.configure_deploy.**validate_open_port**(*v*)

almir.scripts.configure_deploy.**validate_timezone**(*v*)

almir.scripts.parse_console_commands – Parse help commands from bconsole source

almir.scripts.parse_console_commands.**main**()

almir.scripts.parse_console_commands.**parse_console_commands**(*source*)

4.1.7 almir.tests – Tests package

almir.tests.test_functional – Functional tests

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

a