django-business-logic Documentation

Release 0.5.5

Dmitry Kuksinsky

Dec 20, 2019

Contents:

CHAPTER 1

Preface

Processes change more often than technic. Domain Rules are situational and may differ from customer to customer. With diverse code and frequent changes, the code becomes complicated, then the probability of errors increases. One of the optimal strategies to avoid this situation is to isolate client logic at the data level. When this is the case, programmers need only to watch the correct interpretation of the rules, but not each of the rules separately. This reduces the amount of code base needed and reduces the possibility of introducing errors. Another question is the form of the description of these rules takes. A visual graphical representation is optimal for users – this helps to avoid confusion with a variety of conditions, and helps users understand what is written without any effort. Many web applications need to use Domain Specific Languages (DSL) during their lifecycle. Any type of project, from hobby-level to enterprise, may have this requirement. There are many ways to carry out this task, such as doing your own development using internal resources or contractors (you know the cost in both cases), buying a turnkey solution, (really?) or an adapted solution. In all cases, you will also need to be a coach for all DSL users.

Stop now and focus your attention here. The Blockly visual programming library was released in 2012, and since 2014 it has been developed with the support of Google. Code.org, a related project that uses Blockly in its main subsystems, has had hundreds of millions of users of different ages in its Hour of Code project. These users started as young as elementary school, and very quickly learned how to program using the amazing Blockly library. Even top universities teach block-based coding (e.g., Berkeley, Harvard). But Blockly is not just a trainer or a toy. It is suitable for serious tasks: form processing, workflows, reporting, email marketing, bots, tests, surveys, quizzes and many other purposes.

The django-business-logic library tries to utilise the block programming approach to delegate programming tasks to non-programmers by implementing a framework for creating and working with Visual Domain Specific Languages.

The library comes with:

- pretty easy integration: minor changes in INSTALLED_APPS and root urlconf
- support of many types of built-in django model fields: all numeric, boolean, string, foreign keys
- logical division into interface/program/version: firstly define an “interface” object which is just a named set of django models and its fields; after you can create one or more named “programs” which implements the given interface and than release any number of the program’s “versions”
- easy invocation injecting of visually edited code: a few strings in your code, at any place - a view controller, signal handler, asynchronous task, no matter the monolith or microservice

- dropdown controls with custom model object list; you need only register your custom models in the standard django admin interface
- ability to easily integrate and make accessible to the visual editor two types of python functions: imported (from python built-in, third-party or your own module) and editable through django admin
- easy to understand Blockly-based log viewer with the ability to inspect intermediate values

Do not forget that this library is not designed with professional developers in mind, so the interface may look ugly and awkward in a professional developer's eyes. Try to disengage from this feeling. Think like a human who gets panicked when seeing (our lovely) code with obscure brackets, odd commas, strange arrows, and so on. We, the development part of the team, can help our teammates from other departments to do more. Let's bring the spirit of our successful sprints and fun development rooms to colleagues from management, marketing, sales, logistics and many others.

CHAPTER 2

Library home

https://github.com/dgk/django-business-logic/

CHAPTER 3

Documentation

https://django-business-logic.readthedocs.io/en/latest/

CHAPTER 4

Support us

Any feedback and github stars would be greatly appreciated.

Please feel free to ask any questions in the gitter chat

If you are familiar with react/mobx and want to help to build django-business-logic library (next generation frontend code is here), please contact me dgk@dgk.su.

CHAPTER 5

License

MIT license

5.1 Caveats

Warning: Please note that this software is still Alpha/Beta quality and is not recommended for production use. The library is currently in active development and any API may be changed. Migration history for models is not supported now. Some internal objects are world writable!

Note: If you want to use this library before it have implemented authenticating/authorization subsystem you can use nginx/apache/your web server .htaccess-like rules by limiting access to the `/business_logic/` url path.

5.2 Demo application

5.2.1 Description

Demo application is a simple bookshelf with the following models:

class sites.dev.books.models.**Publisher**(*id*, *name*, *rank*)

class sites.dev.books.models.**Author**(*id*, *first_name*, *last_name*, *rank*)

class sites.dev.books.models.**Book**(*id*, *title*, *publisher*, *publication_date*, *price*)

On each entering to single book view the following python code executed:

```
# sites/dev/books/views.py
class BookDetail(generic.DetailView):
    model = Book

    def get_object(self, queryset=None):
        book = super(BookDetail, self).get_object(queryset)
        program = Program.objects.get(code='on_book_view')
        version = program.versions.order_by('id').last()
        version.execute(context=Context(debug=True, log=True), book=book)
        book.publisher.save()

        return book
```

You can see screenshots now: *Demo code editing* and *Demo code execution log*

5.2.2 Hosted demo

Hosted demo site is available here: https://django-business-logic-demo.dev.dgk.su/ . The database is reset every hour. You can login into django admin interface with username `test` and password `test`.

5.2.3 Running using Docker

You can run demo app locally using docker:

```
docker build . -t django-business-logic-demo
docker run --rm -it -p 8000:8000 django-business-logic-demo
```

or using docker-compose:

```
docker-compose up
```

Also you can use prebuilt image:

```
docker run --rm -it -p 8000:8000 dgksu/django-business-logic:demo
```

Now you can login into django admin interface http://localhost:8000/admin/ with username `test` and password `test`.

5.2.4 Deployment to Heroku

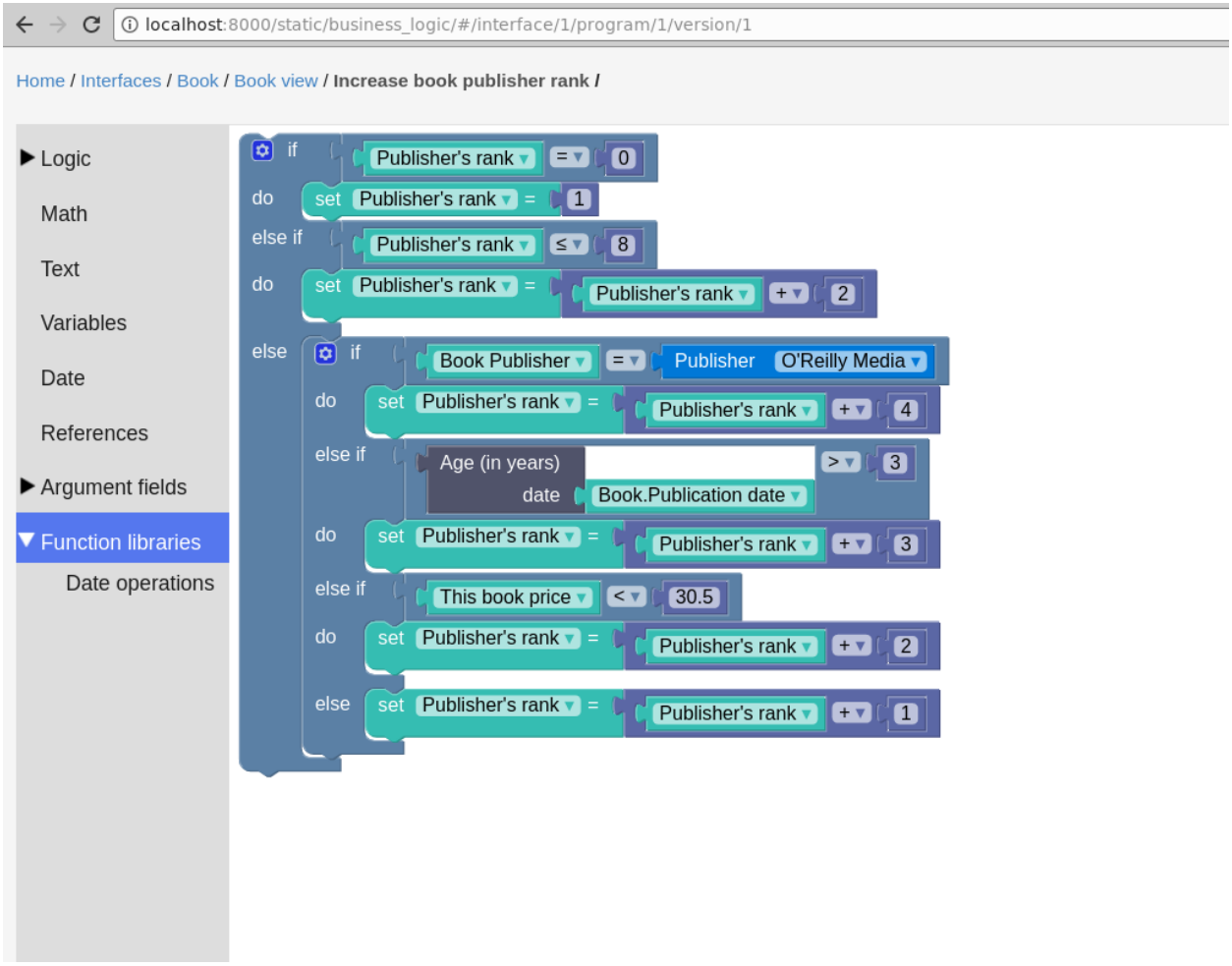
You can deploy demo app directly to Heroku to see the app live. Just click the button below. This will walk you through getting this app up and running on Heroku in minutes.

5.2.5 Running using local files

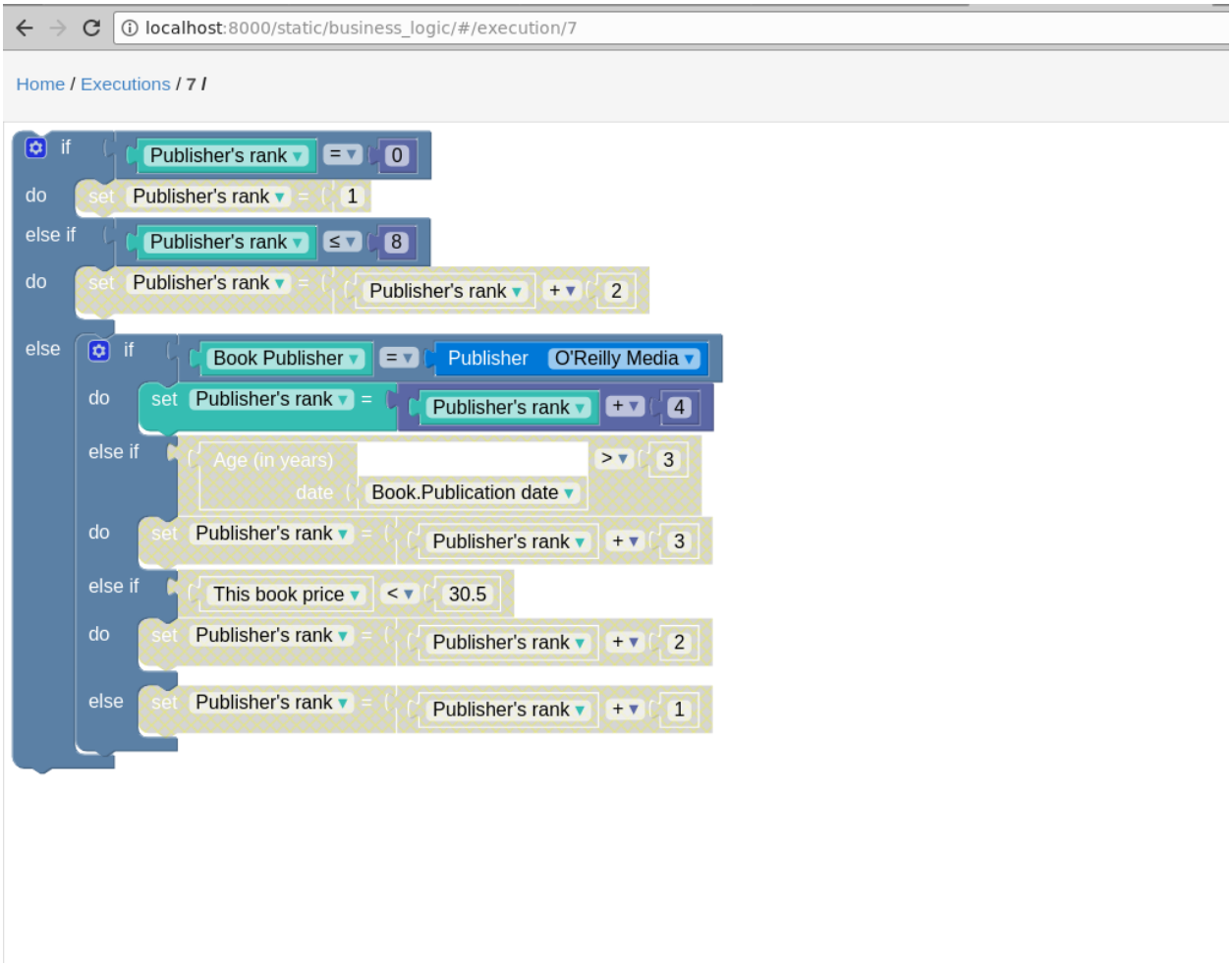
See *Backend development environment*

5.3 Screenshots

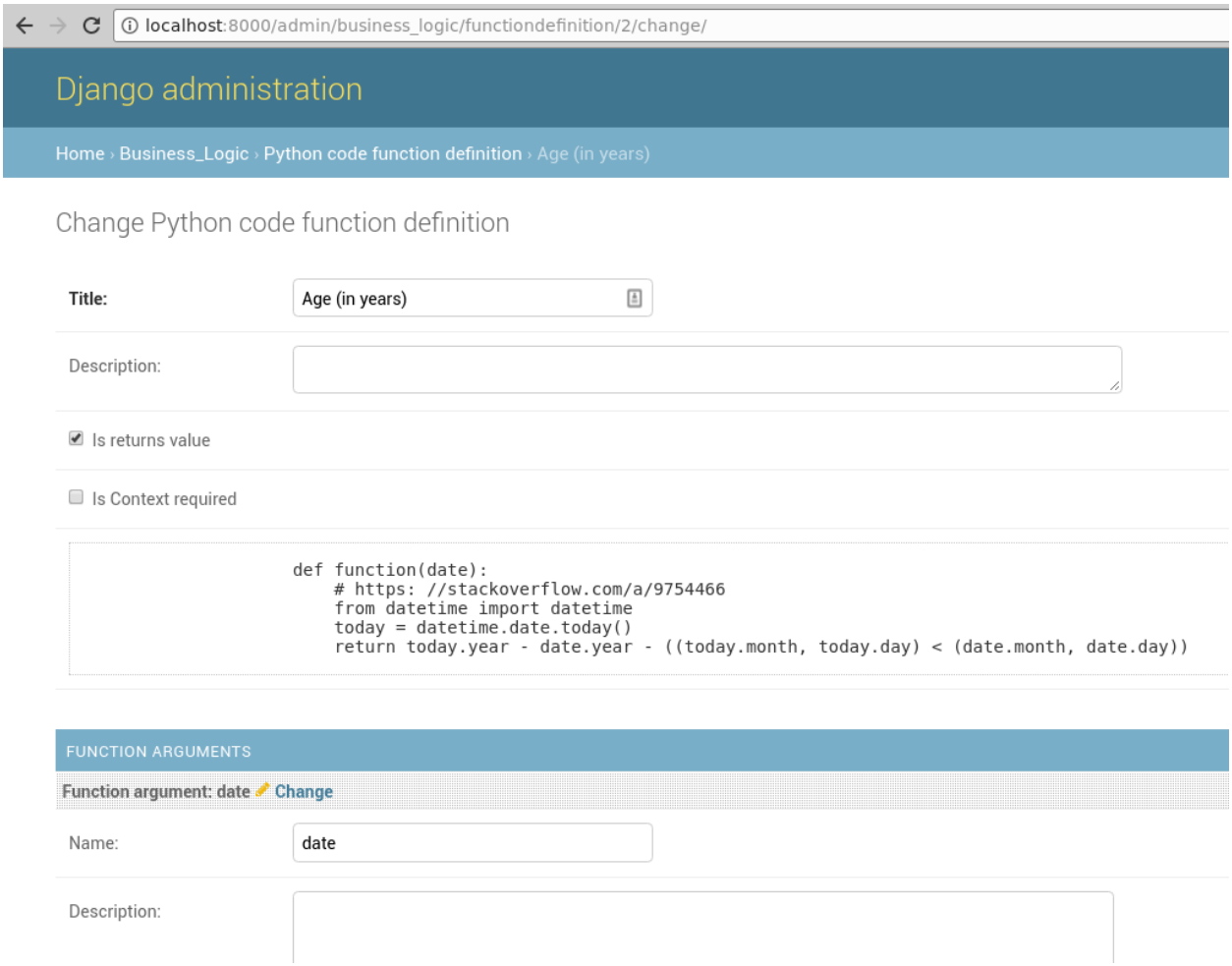
5.3.1 Demo code editing



5.3.2 Demo code execution log



5.3.3 Sample python function



5.4 Requirements

This library requires the following:

- Python (2.7, 3.4, 3.5, 3.6, 3.7)
- Django (1.11, 2.0, 2.1) (for Django 1.8, 1.9, 1.10 try use 0.4.13 version)
- django-rest-framework 3.8+

5.5 Installation

5.5.1 Library installation

There are a few different ways you can install django-business-logic:

- Use pip: `pip install -U django-business-logic`
- Download a zipfile from the releases page and install it.

- Checkout the source: `git clone git://github.com/dgk/django-business-logic.git` and install it yourself by entering `python setup.py install`.

5.5.2 Server setup

Edit `settings.py` and add following apps to your `INSTALLED_APPS`

```
# settings.py
INSTALLED_APPS = (
    # ...
    'django.contrib.contenttypes',

    'ace_overlay', # optional, for comfortable python functions editing
    'adminsortable2',
    'nested_inline',
    'polymorphic',
    'rest_framework', # optional, provided browsable API for this library handy␣
↪development
    'django_filters', # ^^ same

    'business_logic',
    # ...
)
```

Edit `urls.py` and include required urls

```
# urls.py
urlpatterns = (
  # ...
  url('^business-logic/', include('business_logic.urls')),
  url('^nested_admin/', include('nested_admin.urls')),
  # ...
)
```

Make migrations

```
python manage.py migrate
```

Collect static files

```
python manage.py collectstatic
```

5.6 Configuration

5.6.1 Administrative setup

First you should define one or more ProgramInterface objects via django admin interface (http://localhost:8000/admin/business_logic/programinterface/).

Each ProgramInterface must contain one or more ProgramArgument objects. The ProgramArgument object represents one instance of django.db.Model derived class specified as django.contrib.contentypes.ContentType instance (e.g. for your custom Order model) and his keyword argumet name (e.g. `order`).

Each ProgramArgument object must contain one or more ProgramArgumentField which represents one field of django model (e.g. `sum` for Order object or `delivery_address.city` for `city` field nested into Order DeliveryAddress model).

If you want to use system-wide references (e.g. your custom City or ProductCategory django model) and you define represented ProgramArgumentField you should register referenced model via django admin interface(http://localhost:8000/admin/business_logic/referencedescriptor/).

Next create one or more Program objects which must implements described ProgramInterface(e.g. named "On Order create" with "on_order_create" `code` field for programmatic access)

5.6.2 Invocation injecting

You may inject execution engine call at arbitrary place of your code, such as custom `form.save()`, `model.post_save()` methods, any `django.dispatch.Signal` handler or so on. Just instantiate appropriate `ProgramVersion` object and pass kwargs described in the `ProgramInterface` admin page to its `execute()` method. E.g.:

```
from django.views.generic.edit import CreateView

from business_logic.models import Program

class OrderCreate(CreateView):
    def form_valid(self, form):
        order = form.save()
        program = Program.objects.get(code="on_order_create")
        program_version = program.versions.order_by("id").last()
        program_version.execute(order=order)
```

The *business_logic.models.ProgramVersion.execute()* method can accept instance of *business_logic.models.Context* object. If this parameter omitted `execute()` method creates new instance of *business_logic.models.Context* with specified by *business_logic.models.ExecutionEnvironment* parameters or with default parameters.

It can be initialized by the following parameters:

- `log` (boolean, default - `False`) - create execution log
- `debug` (boolean, default - `False`) - create special *business_logic.models.Execution* object contained:
  - all variables passed to `execute()` method
  - *business_logic.models.ProgramVersion* object
  - start and end execution times
  - root of log objects if its created

The `ProgramVersion.execute()` method returns the Context instance.

5.7 Visual programming

After setup you can go to web editor interface (http://localhost:8000/static/business_logic/index.html), choose program interface, program, create and start editing your first program version.

5.8 Architecture

Internally program code is stored as special django models such as `NumberConstant` (see *Constants*) , `IfStatement`, `Assignment` and so forth. Structure of syntax tree is held by class *Node* derived from treebeard.NS_Node. Operators and operands are linked with `Node` objects through django contenttypes system. Other details are briefly described in *Administrative setup* and *Invocation injecting* sections

5.8.1 Program entities

ProgramInterface

class business_logic.models.**ProgramInterface**(**args*, ***kwargs*)

Determines interface for *business_logic.models.Program*. Should be configured by adding one or more *business_logic.models.ProgramArgument*. Can hold link to *business_logic.models.ExecutionEnvironment*.

title

human-readable name

Type str

code

machine-readable code for programmatic access

Type str

environment

execution environment, can be empty

Type *business_logic.models.ExecutionEnvironment*

programs

queryset of child Program

Type *business_logic.models.Program*

ProgramArgument

class business_logic.models.**ProgramArgument**(*id*, *program_interface*, *name*, *content_type*, *variable_definition*)

ProgramArgumentField

class business_logic.models.**ProgramArgumentField**(**args*, ***kwargs*)

program_argument

argument of *business_logic.models.ProgramInterface*

Type *business_logic.models.ProgramArgument*

name

name of the field, can include dots for nested fields

Type str

title
human-readable name

variable_definition
definition for variable

Type *business_logic.models.VariableDefinition*

Program

class business_logic.models.**Program**(**args*, ***kwargs*)
Implements *business_logic.models.ProgramInterface*. Can hold link to *business_logic.models.ExecutionEnvironment*.

title
human-readable name

Type str

code
machine-readable code for programmatic access

Type str

environment
execution environment, can be empty

Type *business_logic.models.ExecutionEnvironment*

program_interface
implemented interface

Type *business_logic.models.ProgramInterface*

versions
queryset of child ProgramVersion

Type *business_logic.models.ProgramVersion*

ProgramVersion

class business_logic.models.**ProgramVersion**(**args*, ***kwargs*)
Acts as version of *business_logic.models.Program*. Main holder of visually editable code. Can hold link to *business_logic.models.ExecutionEnvironment*.

title
human-readable name

Type str

description
human-readable description

Type str

environment
execution environment, can be empty

Type *business_logic.models.ExecutionEnvironment*

is_default
default=False, can be used for choosing suitable *ProgramVersion*. Only one *ProgramVersion* for given *:class:'business_logic.models.Program* can have this field value as *True*

Type bool

entry_point
entry point of visually editable code

Type *business_logic.models.Node*

program
parent Program

Type *business_logic.models.Program*

copy(*new_title*)
Creates a copy of self with given title. Suitable for "save as" actions.

Parameters **new_title** – *str* new title

Returns new *business_logic.models.ProgramVersion* instance

execute(*context=None*, ***kwargs*)
Main function for program execution

Parameters

- **context** (*business_logic.models.Context*, optional) – Context instance
- ****kwargs** – program arguments

Returns Context instance

Return type *business_logic.models.Context*

Raises

- KeyError – If any of program argument omitted
- AssertionError – if any of program argument have incorrect type, if passed unregistered program argument

---

Todo:

- create own exceptions instead AssertionError/KeyError/etc

See also:

- *business_logic.models.Context*
- *business_logic.models.ExecutionEnvironment*

5.8.2 Node

class business_logic.models.**Node**(**args*, ***kwargs*)
Derived from treebeard.NS_Node. Holds structure of syntax tree. All objects are linked using the django contenttypes framework. Interprets code using the *business_logic.models.Node.interpret()* method. Can acts as parent of code block if not contains content_object. Can contains comment.

See also:

- *business_logic.models.NodeCache*
- *business_logic.models.NodeCacheHolder*

static ensure_content_object_saved(***kwargs*)
Saves content_object if needed.

classmethod add_root(***kwargs*)
Adds a root node to the tree. Saves content_objects if needed. :param **kwargs: kwargs for *business_logic.models.Node* constructor

Returns created root node

Return type *business_logic.models.Node*

add_child(***kwargs*)
Adds a child to the node. Saves content_objects if needed.

Parameters ****kwargs** – kwargs for *business_logic.models.Node* constructor

Returns created node

Return type *business_logic.models.Node*

delete()
Removes a node and all it's descendants, and content_objects if needed.

clone()
Creates a clone of entire tree starting from self

Returns root node of cloned tree

Return type *business_logic.models.Node*

interpret(*ctx*)
Interprets the held code.

Parameters **ctx** (*business_logic.models.Context*) – execution context

Returns interpreted value

pprint()
Prints entire tree starting from self to stdout.

Utility function for development purposes.

class business_logic.models.**NodeCache**
Creates cache with preloaded content objects for entire tree on first call of get_children().

Uses *1 + n* SQL queries, where n is count of used content types.

get_children(*node*)
Returns cached child nodes

Parameters **node** (*business_logic.models.Node*) – parent node

Returns list of *business_logic.models.Node*

class business_logic.models.**NodeCacheHolder**
Implements get_children() function using *business_logic.models.NodeCache*

get_children(*node*)
Returns cached child nodes

Parameters **node** (*business_logic.models.Node*) – parent node

Returns list of *business_logic.models.Node*

class business_logic.models.**NodeVisitor**
Utility class for tree traversal.

Derived class should implement the *business_logic.models.NodeVisitor.visit()* method. Traversal is made by executing *business_logic.models.NodeVisitor.preorder()* or *business_logic.models.NodeVisitor.postorder()* method.

Examples

- *business_logic.models.Node.clone()*
- *business_logic.models.Node.pprint()*

visit(*node*, **args*, ***kwargs*)
Main method which should be realised in derived classes

Parameters

- **node** (*business_logic.models.Node*) – currently processed node
- ***args** – args passed to *business_logic.models.NodeVisitor.preorder()* or *business_logic.models.NodeVisitor.postorder()*
- ****kwargs** – kwargs passed to *business_logic.models.NodeVisitor.preorder()* or *business_logic.models.NodeVisitor.postorder()*

preorder(*node*, **args*, ***kwargs*)
Tree traversal from top to bottom.

Parameters

- **node** (*business_logic.models.Node*) – node for starting tree traversal
- ***args** – arbitrary args which should be passed to *business_logic.models.NodeVisitor.visit()*
- ****kwargs** – arbitrary kwargs which should be passed to *business_logic.models.NodeVisitor.visit()*

postorder(*node*, **args*, ***kwargs*)
Tree traversal from bottom to top.

Parameters

- **node** (*business_logic.models.Node*) – node for starting tree traversal
- ***args** – arbitrary args which should be passed to *business_logic.models.NodeVisitor.visit()*
- ****kwargs** – arbitrary kwargs which should be passed to *business_logic.models.NodeVisitor.visit()*

5.8.3 Operators

class business_logic.models.**BinaryOperator**(**args*, ***kwargs*)
Implements binary operations.

Supported operators are:

- +
- -
- *
- /
- ^
- %
- &
- |
- ==
- !=
- >
- >=
- <
- <=
- in

class business_logic.models.**UnaryOperator**(**args*, ***kwargs*)
Implements unary operations.

Supported operators are:

- -
- not
- neg
- abs

class business_logic.models.**Assignment**(**args*, ***kwargs*)
Assignment value to variable

interpret(*ctx*, *lhs*, *rhs*)
Interpret assignment

Parameters

- **ctx** (*business_logic.models.Context*) – execution context
- **lhs** (*business_logic.models.Variable*) – assignment variable
- **rhs** – right hand side value

Returns rhs value

5.8.4 Operands

Constants

class business_logic.models.**Constant**(**args*, ***kwargs*)
Abstract class for constants storing

class business_logic.models.**NumberConstant**(*id*, *value*)

class business_logic.models.**StringConstant**(*id*, *value*)

class business_logic.models.**BooleanConstant**(*id*, *value*)

class business_logic.models.**DateConstant**(*id*, *value*)

References

class business_logic.models.**ReferenceConstant**(**args*, ***kwargs*)
A special type of constant that uses the value in the first child node of the node that stores it

class business_logic.models.**ReferenceDescriptor**(*id*, *content_type*, *title*, *search_fields*, *name_field*)

Variables

class business_logic.models.**VariableDefinition**(*id*, *name*)

class business_logic.models.**Variable**(**args*, ***kwargs*)

class business_logic.models.Variable.**Undefined**

5.8.5 Statements

class business_logic.models.**IfStatement**(*id*)

Not fully implemented:

class business_logic.models.**StopInterpretation**(*id*)

class business_logic.models.**BreakLoop**(*id*)

5.8.6 Execution environment

class business_logic.models.**ExecutionEnvironment**(**args*, ***kwargs*)
Environment of execution.

Can be linked to any of *business_logic.models.ProgramInterface* -> *business_logic.models.Program* -> *business_logic.models.ProgramVersion* chain.

Contains list of *business_logic.models.FunctionLibrary* and defines parameters for *business_logic.models.Context* creation.

title
human-readable name

Type str

description
description

Type str

libraries
list of *business_logic.models.FunctionLibrary* available to execution

debug
default=False

Type bool

log
default=False

Type bool

cache
default=True

Type bool

exception_handling_policy

Type *business_logic.config.ExceptionHandlingPolicy*

5.8.7 Context

class business_logic.models.**Context**(***kwargs*)

config

Type *business_logic.config.ContextConfig*

get_variable(*variable_definition*)
Get variable value in current context

Parameters **variable_definition** (*business_logic.models.VariableDefinition*) – definition of variable

Returns variable value

set_variable(*variable_definition*, *value*)
Set variable value in current context

Parameters

- **variable_definition** (*business_logic.models.VariableDefinition*) – definition of variable
- **value** – variable value

class business_logic.config.**ContextConfig**(***kwargs*)
Stores configuration of *business_logic.models.Context*

Parameters **kwargs** (*object*) – overrides default configuration values

Raises TypeError

defaults = {'cache': True, 'debug': False, 'exception_handling_policy': 'INTERRUPT'
default configuration values

Type object

class business_logic.config.**ExceptionHandlingPolicy**
Enumeration of names of exception handling policies

IGNORE = 'IGNORE'
Ignore exception and continue execution

INTERRUPT = 'INTERRUPT'
Interrupt execution, this is default behaviour

RAISE = 'RAISE'
Re-raise exception

5.8.8 Functions

Imported functions

class business_logic.models.**PythonModuleFunctionDefinition**(*id*, *polymorphic_ctype*, *title*, *description*, *is_context_required*, *is_returns_value*, *functiondefinition_ptr*, *module*, *function*)

Editable functions

class business_logic.models.**PythonCodeFunctionDefinition**(**args*, ***kwargs*)

Todo:

- re-raise exception
- rewrite block with eval() to more safe

Function libraries

class business_logic.models.**FunctionLibrary**(*id*, *title*)

5.8.9 Logging

class business_logic.models.**Logger**
Processed log creation during calling the *business_logic.models.ProgramVersion.execute()* method. Will work only if `Context.config.log` flag is set to `True`.

See also:

- *business_logic.config.ContextConfig*
- *business_logic.models.Context*
- *Signals*

class business_logic.models.**LogEntry**(**args*, ***kwargs*)
Derived from treebeard.AL_Node. Stores single *business_logic.models.Node* interpretation event. Will be created only if `Context.config.log` flag is set to `True`.

node
currently interpreted node

Type *business_logic.models.Node*

previous_value
value before interpretation (actual for *business_logic.models.Variable*)

current_value
value after interpretation

exception
exception if it raised during interpretation.

Type *business_logic.models.ExceptionLog*

See also:

- *business_logic.config.ContextConfig*
- *business_logic.models.Context*

class business_logic.models.**ExceptionLog**(**args*, ***kwargs*)
Stores information about exception raised during node interpretation.

log_entry
parent LogEntry

Type *business_logic.models.LogEntry*

module
exception module

Type str

type
exception type

Type str

message
exception message

Type str

traceback
traceback

Type str

class business_logic.models.**Execution**(**args*, ***kwargs*)
Stores information about calling the *business_logic.models.ProgramVersion.execute()* method. Will be created only if `Context.config.debug` flag is set to `True`.

program_version
executed ProgramVersion

Type *business_logic.models.ProgramVersion*

arguments
arguments of execution

Type list of *business_logic.models.ExecutionArgument*

start_time
start execution time

Type datetime

finish_time
finish execution time

Type datetime

log
root node of LogEntries

Type *business_logic.models.LogEntry*

See also:

- *business_logic.config.ContextConfig*
- *business_logic.models.Context*

class business_logic.models.**ExecutionArgument**(**args*, ***kwargs*)
Stores information about argument passed to *business_logic.models.ProgramVersion.execute()*.

execution
parent Execution object

Type *business_logic.models.Execution*

program_argument
parent ProgramArgument object

Type *business_logic.models.ProgramArgument*

content_object
passed argument

Type django.db.models.Model

5.8.10 Signals

business_logic.signals.**block_interpret_enter = <django.dispatch.dispatcher.Signal object>**
Fired on entering to code block interpretation

business_logic.signals.**block_interpret_leave = <django.dispatch.dispatcher.Signal object>**
Fired on leaving code block interpretation

business_logic.signals.**statement_interpret_enter = <django.dispatch.dispatcher.Signal objec**
Fired on entering statement interpretation

business_logic.signals.**statement_interpret_leave = <django.dispatch.dispatcher.Signal objec**
Fired on leaving statement interpretation

business_logic.signals.**interpret_enter = <django.dispatch.dispatcher.Signal object>**
Fired on entering code interpretation

business_logic.signals.**interpret_leave = <django.dispatch.dispatcher.Signal object>**
Fired on leaving code interpretation

business_logic.signals.**interpret_exception = <django.dispatch.dispatcher.Signal object>**
Fired on interpretation exception

5.8.11 Exceptions

exception business_logic.exceptions.**InterpretationException**
Wraps python exception raised during running *business_logic.models.Node.interpret()*.

exception business_logic.exceptions.**StopInterpretationException**
Exception for stop interpretation. This feature not implemented.

See also:

- *business_logic.models.StopInterpretation*

exception business_logic.exceptions.**BreakLoopException**
Exception for breaking loops. This feature not implemented.

See also:

- *business_logic.models.BreakLoop*

5.9 Library Development

5.9.1 Backend development environment

virtualenv

Virtualenv is probably what you want to use during development. Once you have virtualenv installed, just fire up a shell and create your own environment.

```
virtualenv venv
source venv/bin/activate
pip install -r requirements.dev.txt
```

Installing test data

```
python manage.py migrate
python manage.py loaddata sites/dev/fixtures/data.json
```

Running backend dev server

```
python manage.py runserver
```

An instance of django dev server will be listening on http://localhost:8000/ . Now you can login into django admin interface http://localhost:8000/admin/ with username `test` and password `test`.

5.9.2 Frontend development environment

Fronted source files located under the `frontend` folder.

```
cd frontend
```

Installing dependencies

```
npm install
```

Running webpack dev server

```
npm run server:dev:hmr
```

Now webpack dev server will be listening on http://localhost:3000/ .

Building frontend files

```
npm run build:prod
```

5.9.3 Running tests

Running backend tests

```
python manage.py test
```

Test it all

You need to know at least one command; the one that runs all the tests:

```
tox
```

5.10 Roadmap

- Frontend refactoring
- Lists and Loops
- Implicit strong static typing
- Authorization
- Two-dimensional constant tables (matrices)
- Pure python code generation for execution speedup
- Visually editable functions and its libraries
- Code sharing

5.10.1 Todo

Todo:

- re-raise exception
- rewrite block with eval() to more safe

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/django-business-logic/checkouts/latest/business_logic/models/function.py:docstring of business_logic.models.PythonCodeFunctionDefinition, line 1.)

Todo:

- create own exceptions instead AssertionError/KeyError/etc

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/django-business-logic/checkouts/latest/business_logic/models/program.py:docstring of business_logic.models.ProgramVersion.execute, line 14.)

5.11 Credits

Many thanks to:

- contributors of this library
- all folks from Insurance Technologies LLC (b2bpolis.ru, @b2bpolis), St.Petersburg, Russia and personally to its CEO Roman Kurdo
- authors of all used opensource libraries

5.12 License

Python Module Index

b

s

Index

R

S

T

U

V