link.utils Documentation

Release 2.0

David Delassus

October 10, 2016

Contents

link.utils is an utility library for other *link* frameworks.

Check out the source code on Github.

CHAPTER 1

Installation

```
pip install link.utils
```

CHAPTER 2

Contents

2.1 Tutorial

2.1.1 Filtering and mangling

The filter spec is based on MongoDB:

```
from link.utils.filter import Filter

spec = {
    # ...
}
f = Filter(spec)

if f.match(dictionary):
    print 'match'
```

And the mangling spec is also based on MongoDB:

```
from link.utils.filter import Mangle

spec = {
    # ...
}

m = Mangle(spec)
newdict = m(dictionary)
```

2.1.2 Logging

The logging module sets a new logging class, it just needs to be imported. Hopefully, the library `b3j0f.conf` provides a way to do that automatically:

Edit the file `$B3J0F_CONF_DIR/b3j0fconf-configurable.conf`:

```
{
  "CONFIGURABLE": {
    "modules": [
      "link.utils.log"
    ]
  }
}
```

Then, configure the new logging class by editing the file `$B3J0F_CONF_DIR/link/utils/logging.cong`:

```
{
  "LOGGING": {
    "log_format": "[\%(asctime)s] [\%(levelname)s] [\%(name)s] \%(message)s",
    "log_level": "INFO",
    "log_filter": {
      "name": {"$regex": "^link\..*"}
    }
  }
}
```

NB: The `log_filter` option is a MongoDB filter used to filter logging records.

Finally, just use the `logging` module as usual.

2.1.3 Logging Records

For more information about what a logging record is, click here.

A record is transformed into a `dict` for filtering, it validates the following JSON schema:

```
{
  "$schema": "http://json-schema.org/schema#",

  "type": "object",
  "properties": {
    "name": {
      "title": "record.name",
      "description": "Logging record name attribute",
      "type": "string"
    },
    "level": {
      "title": "record.level",
      "description": "Logging record level attribute",
      "type": "integer"
    },
    "pathname": {
      "title": "record.pathname",
      "description": "Logging record pathname attribute",
      "type": "string"
    },
    "lineno": {
      "title": "record.lineno",
      "description": "Logging record lineno attribute",
      "type": "integer"
    },
    "msg": {
      "title": "record.msg % record.args",
      "description": "Logging record msg attribute formatted with args attribute",
      "type": "string"
    },
    "func": {
      "title": "record.func",
      "description": "Logging record func attribute",
      "type": "string"
    },
```

```
    "sinfo": {
      "title": "record.sinfo",
      "description": "Logging record sinfo attribute (null if Python 2)",
      "$oneOf": [
         {"type": "string"},
         {"type": "null"}
      ]
    },
    "exc_info": {
      "title": "record.exc_info",
      "description": "Logging record exc_info attribute",
      "$oneOf": [
        {
          "type": "object",
          "properties": {
            "type": {
              "title": "record.exc_info[0].__name__",
              "description": "Exception's name",
              "type": "string"
            },
            "msg": {
              "title": "str(record.exc_info[1])",
              "description": "Exception's value",
              "type": "string"
            },
            "traceback": {
              "title": "''.join(traceback.format_tb(record.exc_info[2]))",
              "description": "Exception's traceback",
              "type": "string"
            }
          }
        },
        {"type": "null"}
      ]
    }
  }
}
```

2.1.4 Code generation

Based on the library Grako, parser code generation from BNF is provided with:

```
from link.utils.grammar import codegenerator


with open('grammar.bnf') as f:
    module = codegenerator('mydsl', 'MyDSL', f.read())

parser = module.MyDSLParser()

with open('mycode') as f:
    # 'start' is the default rule name used to start parsing
    model = parser.parse(f.read(), rule_name='start')
```

When using the `NodeWalker` class to traverse the model (built with the `ModelBuilderSemantics` class), those two functions are usefull:

```
from link.utils.grammar import codegenerator, adopt_children, find_ancestor
from grako.model import ModelBuilderSemantics


with open('grammar.bnf') as f:
    module = codegenerator('mydsl', 'MyDSL', f.read())

parser = module.MyDSLParser(semantics)

with open('mycode') as f:
    # 'start' is the default rule name used to start parsing
    model = parser.parse(f.read(), rule_name='start')
```

Use this before calling the NodeWalker in order to have nodes parent member set:

```
adopt_children(model._ast, parent=model)
```

When traversing the model, you may need to get informations about a parent node:

pnode = find_ancestor(node, 'ParentNode')

if pnode is not None: # parent node was found

2.2 API documentation

2.2.1 link.utils package

Submodules

link.utils.filter module

class link.utils.filter.**Filter**(*rule*, **args*, ***kwargs*)

Bases: object

Apply MongoDB filter rule on dictionary.

handle_all_field(*key*, *rule*, *obj*)

Handle $all operator.

Parameters

- **key** (*str*) – key to check in dictionary
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if field match all values

Return type boolean

handle_and(*key*, *rule*, *obj*)

Handle $and operator.

Parameters

- **key** (*str*) – key to check in dictionary (unused)
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if dictionary match all sub-filters

Return type boolean

handle_field(*key*, *rule*, *obj*)
Handle filter on field.

Parameters

- **key** (*str*) – key to check in dictionary
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if dictionary match

Return type boolean

handle_field_cond(*key*, *rule*, *obj*, *cond*)
Handle comparison operators.

Parameters

- **key** (*str*) – key to check in dictionary
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check
- **cond** (*callable*) – comparison operator

Returns True if field match

Return type boolean

handle_field_exists(*key*, *rule*, *obj*)
Handle `$exists` operator.

Parameters

- **key** (*str*) – key to check in dictionary
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if field is in dictionary

Return type boolean

handle_field_regex(*key*, *pattern*, *obj*, *opts=None*)
Handle `$regex` operator.

Parameters

- **key** (*str*) – key to check in dictionary
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if field match regex

Return type boolean

handle_field_rule(*key*, *rule*, *obj*)
Handle other operators.

Parameters

- **key** (*str*) – key to check in dictionary
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if field match

Return type boolean

handle_in_field(*key*, *rule*, *obj*)
Handle `$in` operator.

Parameters

- **key** (*str*) – key to check in dictionary
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if field match at least one value

Return type boolean

handle_nor(*key*, *rule*, *obj*)
Handle `$nor` operator.

Parameters

- **key** (*str*) – key to check in dictionary (unused)
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if dictionary doesn't match any sub-filters

Return type boolean

handle_or(*key*, *rule*, *obj*)
Handle `$or` operator.

Parameters

- **key** (*str*) – key to check in dictionary (unused)
- **rule** (*dict*) – MongoDB sub-filter
- **obj** (*dict*) – dictionary to check

Returns True if dictionary match at least one sub-filter

Return type boolean

match(*obj*)
Check if dictionary match the MongoDB filter.

Parameters **obj** (*dict*) – dictionary to check

Returns True if dictionary match

Return type boolean

class link.utils.filter.**Mangle**(*rule*, **args*, ***kwargs*)
Bases: object

Apply MongoDB update spec on dictionary.

addToSet(*rule*, *obj*)
Handle `$addToSet` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

bit(*rule*, *obj*)
Handle `$bit` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

currentDate(*rule*, *obj*)
Handle `$currentDate` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

inc(*rule*, *obj*)
Handle `$inc` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

max(*rule*, *obj*)
Handle `$max` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

min(*rule*, *obj*)
Handle `$min` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

mul(*rule*, *obj*)
Handle `$mul` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

pop(*rule*, *obj*)
Handle `$pop` operator.

Parameters

- **rule** (*dict*) – sub-spec

- **obj** (*dict*) – dictionary to apply sub-rule on

pull(*rule*, *obj*)
Handle `$pull` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

pullAll(*rule*, *obj*)
Handle `$pullAll` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

push(*rule*, *obj*)
Handle `$push` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

rename(*rule*, *obj*)
Handle `$rename` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

set(*rule*, *obj*)
Handle `$set` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

unset(*rule*, *obj*)
Handle `$unset` operator.

Parameters

- **rule** (*dict*) – sub-spec
- **obj** (*dict*) – dictionary to apply sub-rule on

link.utils.filter.**del_field**(*key*, *obj*)
Remove field from dictionary.

Parameters

- **key** (*str*) – path to field in dictionary
- **obj** (*dict*) – dictionary used for search

link.utils.filter.**get_field**(*key*, *obj*)
Get field in dictionary.

Parameters

- **key** (*str*) – path to field in dictionary
- **obj** (*dict*) – dictionary used for search

Returns field value

link.utils.filter.**set_field**(*key*, *obj*, *val*)
et field in dictionary.

Parameters

- **key** (*str*) – path to field in dictionary
- **obj** (*dict*) – dictionary used for search
- **val** – value to set

link.utils.log module

class link.utils.log.**ConfigurableLogger**(*name*)
Bases: logging.Logger

Configurable logger.

log_filter

log_format

log_level

class link.utils.log.**LogFilter**(*name=''*)
Bases: logging.Filter

Filter log records using link.utils.filter.Filter.

filter(*record*)

log_filter

link.utils.log.**logrecord_to_dict**(*record*)
Transform a LogRecord object into a dict.

Parameters **record** (*LogRecord*) – record to transform

Returns record as a JSON serializable dict

Return type dict

link.utils.grammar module

link.utils.grammar.**codegenerator**(*modname*, *prefix*, *grammar*)
Parse grammar model and generate Python code allowing to parse it.

Example:

```
with open('grammar.bnf') as f:
    module = codegenerator('mydsl', 'MyDSL', f.read())

assert module.__name__ == 'mydsl'
parser = module.MyDSLParser()
```

Parameters

- **modname** (*str*) – Name of the generated Python module
- **prefix** (*str*) – Prefix used to name the parser
- **grammar** (*str*) – Grammar describing the language to parse

Returns Generated Python module

Return type module

link.utils.grammar.**find_ancestor**(*node*, *classname*)
Find first node's ancestor which match class' name.

Parameters

- **node** (*grako.model.Node*) – Grako Node
- **classname** (*str*) – Class' name

Returns Node's ancestor or None if not found

Return type grako.model.Node

Module contents

CHAPTER 3

Donating

l