pykwalify Documentation

Release 1.6.0

Johan Andersson

January 22, 2017

Contents

1 Usage **3**

2 Examples **5**

3 PyYaml and ruamel.yaml **7**

4 UTF-8 and data encoding **9**

5 Project details **11**

6 The Usage Guide **13**
- 6.1 Basic Usage . . . 13
- 6.2 Validation rules . . . 13
- 6.3 Partial schemas . . . 22
- 6.4 Extensions . . . 23

7 The Community Guide **27**
- 7.1 Testing . . . 27
- 7.2 Upgrading instructions . . . 27
- 7.3 Release Notes . . . 28
- 7.4 Authors . . . 35
- 7.5 Licensing . . . 35

PyKwalify is a open source port of the kwalify lib and specification. The original source code was written in Java but this port is based on Python. The code is open source, and available on github.

YAML/JSON validation library

This framework is a port with alot added functionality of the java version of the framework kwalify that can be found at: http://www.kuwata-lab.com/kwalify/

The source code can be found at: http://sourceforge.net/projects/kwalify/files/kwalify-java/0.5.1/

The schema this library is base and extended from: http://www.kuwata-lab.com/kwalify/ruby/users-guide.01.html#schema

CHAPTER 1

Usage

Create a data file. *Json* and *Yaml* formats are both supported.

```
- foo
- bar
```

Create a schema file with validation rules.

```
type: seq
sequence:
  - type: str
```

Run validation from cli.

```
pykwalify -d data.yaml -s schema.yaml
```

CHAPTER 2

Examples

The documentation describes in detail how each keyword and type works and what is possible in each case.

But there is a lot of real world examples that can be found in the test data/files. It shows alot of examples of how all keywords and types work in practise and in combination with eachother.

The files can be found here and it shows both schema/data combinations that will work and that will fail.

- *tests/files/success/*
- *tests/files/fail/*
- *tests/files/partial_schemas/*

CHAPTER 3

PyYaml and ruamel.yaml

`PyYaml` is the default installed yaml parser and `ruamel.yaml` is possible to install at the same time with the following command

```
pip install 'pykwalify[ruamel]'

# or for development:

pip install -e '.[ruamel]'
```

`ruamel.yaml` will however be used if both is installed becuase it is more up to date and includes the YAML 1.2 specification that `PyYaml` do not support.

`PyYaml` will still be the default parser becuase it is used more and is still considered the default `YAML` parser in the python world.

Depending on how both libraries is developed, this can change in the future in any major update.

CHAPTER 4

UTF-8 and data encoding

If you have problems with unicode values not working properly when running pykwalify on python 2.7.x then try to add this environment variable to your execution and it might help to force UTF-8 encoding on all string objects.

If this do not work please open up a issue with your schema and data that can be used to track down the problem in the source code.

```
PYTHONIOENCODING=UTF-8 pykwalify ...
```

CHAPTER 5

Project details

python support	2.7, 3.3, 3.4, 3.5, 3.6, 3.7
Source	https://github.com/Grokzen/pykwalify
Docs (Latest release)	http://pykwalify.readthedocs.io/en/master/
Docs (Unstable branch)	http://pykwalify.readthedocs.io/en/unstable/
Gitter (Free Chat)	
Changelog	https://github.com/Grokzen/pykwalify/blob/unstable/docs/release-notes.rst
Upgrade instructions	https://github.com/Grokzen/pykwalify/blob/unstable/docs/upgrade-instructions.rst
Issues	https://github.com/Grokzen/pykwalify/issues
Travis (master)	
Travis (unstable)	
Test coverage	
pypi	https://pypi.python.org/pypi/pykwalify/
Open Hub	https://www.openhub.net/p/pykwalify
License	MIT https://github.com/Grokzen/pykwalify/blob/unstable/docs/license.rst
Copyright	Copyright (c) 2013-2017 Johan Andersson
git repo	git clone git@github.com:Grokzen/pykwalify.git
install stable	pip install pykwalify

install dev	`$ git clone git@github.com:Grokzen/pykwalify.git p`
	`$ cd ./pykwalify`
	`$ virtualenv .venv`
	`$ source .venv/bin/activate`
	`$ pip install -r dev-requirements.txt`
	`$ pip install -e .`

required dependencies	docopt >= 0.6.2 python-dateutil >= 2.4.2
supported yml parsers	PyYaml >= 3.11 ruamel.yaml >= 0.11.0

CHAPTER 6

The Usage Guide

6.1 Basic Usage

Create a data `json` or `yaml` file.

```
# Data file (data.yaml)
- foo
- bar
```

Create a schema file with validation rules.

```
# Schema file (schema.yaml)
type: seq
sequence:
  - type: str
```

Run validation from cli.

```
pykwalify -d data.yaml -s schema.yaml
```

Or if you want to run the validation from inside your code directly.

```
from pykwalify.core import Core
c = Core(source_file="data.yaml", schema_files=["schema.yaml"])
c.validate(raise_exception=True)
```

If validation fails then exception will be raised.

6.2 Validation rules

PyKwalify supports all rules implemented by the original kwalify and include many more to extend the specification.

6.2.1 type

A `type` specifies what rules and constraints should be applied to this node in the data structure.

The following types are available:

- **any**
  - Will always be true no matter what the value is, even unimplemented types

- **bool**
  - Only **True/False** validates. Integers or strings like `0` or `1`, `"True"` or `"False"` do not validate for bool
- **date**
  - A string or datetime.date object that follows a date format
- **float**
  - Any object that is a float type, or object that python can interpret as a float with the following python code `float(obj)`. Scientific notation is supported for this type, for example `1e-06`.
- **int**
  - Validates only for integers and not floats
- **mapping or map**
  - Validates only for `dict` objects
- **none**
  - Validates only for `None` values
- **number**
  - Validates if value is **int** or **float**
- **scalar**
  - Validates for all but **seq** or **map**. None values will also fail validation.
- **sequence or seq**
  - Validates for lists
- **str**
  - Validates if value is a python **string** object
- **text**
  - Validates if value is **str** or **number**
- **time**
  - Not yet implemented [NYI]
- **timestamp**
  - Validates for basic timestamp formats

Example

```
# Schema
type: str
```

```
# Data
'Foobar'
```

6.2.2 Mapping

A mapping is validates to the `dict` datastructure.

Aliases

- mapping
- map

The map type is implicitly assumed when mapping or its alias map is present in the rule.

```
# Schema
type: map
mapping:
  key_one:
    type: str
```

```
# Data
key_one: 'bar'
```

The schema below sets the mapping type implicitly and is also a valid schema.

```
# Schema
map:
  key_one:
    type: str
```

There are some constraints which are available only for the map type, and expand its functionality. See the allowempty, regex; (regex-pattern) and matching-rule sections below for details.

By default, map keys specified in the map rule can be omitted unless they have the required constraint explictly set to True.

6.2.3 Sequence

Sequence/list of values with the given type of values.

The sequence type is implicitly assumed when sequence or its alias seq is present in the rule.

Aliases

- sequence
- seq

Example

```
# Schema
type: seq
sequence:
  - type: str
```

```
# Data
- 'Foobar'
- 'Barfoo'
```

The schema below sets the sequence type implicitly and is also a valid schema.

```
# Schema
seq:
  - type: str
```

Multiple list entries is supported to enable validation of different types of data inside the sequence.

Note: The original kwalify specification only allowed one entry in the list. This has been extended in PyKwalify to give more flexibility when validating.

Example

```
# Schema
type: seq
sequence:
  - type: str
  - type: int
```

```
# Data
- 'Foobar'
- 123456
```

Will be valid.

6.2.4 Matching

Multiple subrules can be used within the `sequence` block. It can also be nested to any depth, with subrules constraining list items to be sequences of sequences.

The `matching` constraint can be used when the type is `sequence` to control how the parser handles a list of different subrules for the `sequence` block.

- **any**
  - Each list item must satisfy at least one subrules
- **all**
  - Each list item must satisfy every subrule
- *****
  - At least one list item must satisfy at least one subrule

Example

```
# Schema
type: seq
matching: "any"
sequence:
  - type: str
  - type: seq
    sequence:
      - type: int
```

```
# Data
- - 123
- "foobar"
```

6.2.5 Timestamp

Parse a string or integer to determine if it is a valid unix timestamp.

Timestamps must be above `1` and below `2147483647`.

Parsing is done with python-dateutil. You can see all valid formats in the relevant dateutil documentation.

Example

```
# Schema
type: map
mapping:
  d1:
    type: timestamp
  d2:
    type: timestamp
```

```
# Data
d1: "2015-03-29T18:45:00+00:00"
d2: 2147483647
```

All `datetime` objects will validate as a valid timestamp.

PyYaml can sometimes automatically convert data to `datetime` objects.

6.2.6 Date

Parse a string or datetime object to determine if it is a valid date. Date has multiple valid formats based on what standard you are using.

For example 2016-12-31 or 31-12-16 is both valid formats.

If you want to parse a custom format then you can use the *format* keyword to specify a valid datetime parsing syntax. The valid sytax can be found here python-strptime

Example:

```
# Schema
type: date
```

```
# Data
"2015-12-31"
```

6.2.7 Format

Only valid when using *date* or *datetime* type. It helps to define custom datetime formats if the default formats is not enough.

Define the value as a string or a list with foramts as values that uses the builtin python datetime string formatting language. The syntax can be found here python-strptime

```
# Schema
type: date
format: "%Y-%m-%d"
```

```
# Data
"2015-12-31"
```

6.2.8 Required

If the `required` constraint is set to `True`, the key and its value must be present, otherwise a validation error will be raised.

Default is `False`.

Aliases

- `required`
- `req`

Example

```
# Schema
type: map
mapping:
  key_one:
    type: str
    required: True
```

```
# Data
key_one: foobar
```

6.2.9 Enum

Set of possible elements, the value must be a member of this set.

Object in enum must be a list of items.

Currently only exact case matching is implemented. If you need complex validation you should use `pattern`.

Example

```
# Schema
type: map
mapping:
  blood:
    type: str
    enum: ['A', 'B', 'O', 'AB']
```

```
# Data
blood: AB
```

6.2.10 Pattern

Specifies a regular expression pattern which the value must satisfy.

Uses re.match internally. Pattern works for all scalar types.

For using regex to define possible key names in mapping, see `regex; (regex-pattern)` instead.

Example

```
# Schema
type: map
mapping:
  email:
    type: str
    pattern: .+@.+
```

```
# Data
email: foo@mail.com
```

6.2.11 Range

Range of value between

- `min` or `max`
- `min-ex` or `max-ex`.

For numeric types (`int`, `float` and `number`), the value must be within the specified range, and for non-numeric types (`map`, `seq` and `str`) the length of the `dict/list/string` as given by `len()` must be within the range.

For the data value (or length), x, the range can be specified to test for the following:

- `min` provides an inclusive lower bound, `a <= x`
- `max` provides an inclusive upper bound, `x <= b`
- `min-ex` provides an exclusive lower bound, `a < x`
- `max-ex` provieds an exclusive upper bound, `x < b`

Non-numeric types require non-negative values for the boundaries, since length can not be negative.

Types `bool` and `any` are not compatible with `range`.

Example

```
# Schema
type: map
mapping:
  password:
    type: str
    range:
      min: 8
      max: 16
  age:
    type: int
    range:
      min: 18
      max-ex: 30
```

```
# Data
password: foobar123
age: 25
```

6.2.12 Unique

If unique is set to `True`, then the sequence cannot contain any repeated entries.

The unique constraint can only be set when the type is `seq / sequence`. It has no effect when used with `map / mapping`.

Default is `False`.

Example

```
# Schema
type: seq
sequence:
  - type: str
    unique: True
```

```
# Data
- users
- foo
- admin
```

6.2.13 Allowempty

Only applies to `mapping`.

If `True`, the map can have keys which are not present in the schema, and these can map to anything.

Any keys which **are** specified in the schema must have values which conform to their corresponding constraints, if they are present.

Default is `False`.

Example

```
# Schema
type: map
mapping:
  datasources:
    type: map
    allowempty: True
```

```
# Data
datasources:
  test1: test1.py
  test2: test2.py
```

6.2.14 Regex;(regex-pattern)

Only applies to `mapping` type.

Aliases

- `re;(regex-pattern)`

This is only implemented in `mapping` where a key inside the mapping keyword can implement this `regex;(regex-pattern)` pattern and all keys will be matched against the pattern.

Please note that the regex should be wrapped with `( )` and these parentheses will be removed at runtime.

If a match is found then it will be parsed against the subrules on that key. A single key can be matched against multiple regex rules and the normal map rules.

When defining a regex key, `matching-rule` should also be set to configure the behaviour when using multiple regexes.

Example

```
# Schema
type: map
matching-rule: 'any'
mapping:
  regex;(mi.+):
    type: seq
    sequence:
      - type: str
```

```
  regex;(me.+):
    type: number
```

```
# Data
mic:
  - foo
  - bar
media: 1
```

6.2.15 Matching-rule

Only applies to `mapping`. This enables more finegrained control over how the matching rule should behave when validation regex keys inside mappings.

Currently supported constraint settings are

- **`any`**
  - One or more of the regex must match.
- **`all`**
  - All defined regex must match each key.

Default is `any`.

Example

The following dataset will raise an error because the key `bar2` does not fit all of the regex. If the constraint was instead `matching-rule: all`, the same data would be valid because all the keys in the data match one of the regex formats and associated constraints in the schema.

```
# Schema
type: map
matching-rule: all
mapping:
  regex;([1-2]$):
    type: int
  regex;(^foobar):
    type: int
```

```
# Data
foobar1: 1
foobar2: 2
bar2: 3
```

6.2.16 Name

Name of the schema.

This have no effect on the parsing, but is useful for humans to read.

Example

```
# Schema
name: foobar schema
```

6.2.17 Desc

Description of schema.

This have no effect on the parsing, but is useful for humans to read. Similar to `name`.

Value for desc `MUST` be a string otherwise a `RuleError` will be raised upon usage.

Example

```
# Schema
desc: This schema is very foobar
```

6.2.18 Example

Write a example that can show what values is upported. Or just type any comment into the schema for future reference.

It is possible to use in all levels and places in the schema and have no effect on the parsing, but is useful for humans to read. Similar to `desc`.

Value for `example` `MUST` be a string otherwise a `RuleError` will be raised upon usage.

Example

```
# Schema
example: List of values
type: seq
sequence:
  - type: str
    unique: true
    example: Each value must be unique and a string
```

6.3 Partial schemas

It is possible to create small partial schemas that can be included in other schemas.

This feature do not use any built-in `YAML` or `JSON` linking.

To define a partial schema use the keyword `schema;(schema-id):`. `(schema-id)` name must be globally unique. If collisions is detected then error will be raised.

To use a partial schema use the keyword `include: (schema-id):`. This will work at any place you can specify the keyword `type`. Include directive do not currently work inside a partial schema.

It is possible to define any number of partial schemas in any schema file as long as they are defined at top level of the schema.

For example, this schema contains one partial and the regular schema.

```
# Schema
schema;map_str:
  type: map
  mapping:
    foo:
      type: str

type: seq
sequence:
  - include: map_str
```

```
# Data
- foo: opa
```

6.3.1 schema;(schema-name)

See the `Partial schemas` section for details.

Names must be globally unique.

Example

```
# Schema
schema;list_str:
  type: seq
  sequence:
    - type: str

schema;list_int:
 type: seq
 sequence:
   - type: int
```

6.3.2 Include

Used in `partial schema` system. Includes are lazy and are loaded during parsing / validation.

Example

```
# Schema [barfoo.yaml]
schema;list_str:
  type: seq
  sequence:
    - type: str
```

```
# Schema [foobar.yaml]
include: list_str
```

```
# Data
- foobar
```

6.4 Extensions

It is possible to extend the validation of each of the three basic types, `map` & `seq` & `scalar`.

Extensions can be used to do more complex validation that is not natively supported by the core pykwalify lib.

6.4.1 Loading extensions

There are 2 ways to load extensions into a schema.

First you can specify any `*.py` file via the cli via the `-e FILE` or `--extension FILE` flag. If you would do this when using pykwalify as a library you should pass in a list of files to the `extensions` variable to the `Core` class.

The second way is to specify a list of files in the keyword `extensions` that can only be specified at the top level of the schema. The files can be either relative or absolute.

6.4.2 How custom validation works

Each function defined inside the extension must be defined with a globally unique method name and the following variables

```
def method_name(value, rule_obj, path):
    pass
```

To raise a validation error you can either raise any exception and it will propegate up to the caller or you can return `True` or `False`. Any value/object that will be interpreted as `False` inside a if check will cause a `CoreError` validation error to be raised.

When using a validation function on a `sequence`, the method will be called with the entire list content as the value.

When using a validation function on a `mapping`, the method will be called with the entire dict content as the value.

When using a validation function on any `scalar` type value, the method will be called with the scalar value.

This is a example of how to use extensions inside a simple schema

```
# Schema
extensions:
  - e.py
type: map
func: ext_map
mapping:
  foo:
    type: seq
    func: ext_list
    sequence:
      - type: str
        func: ext_str
```

```
# Data
foo:
  - foo
  - bar
```

This is the extension file named `ext.py` that is located in the same directory as the schema file.

```
# -*- coding: utf-8 -*-
import logging
log = logging.getLogger(__name__)


def ext_str(value, rule_obj, path):
    log.debug("value: %s", value)
    log.debug("rule_obj: %s", rule_obj)
    log.debug("path: %s", path)

    # Either raise some exception that you have defined your self
    # raise AssertionError('Custom assertion error in jinja_function()')

    # Or you should return True/False that will tell if it validated
    return True
```

```
def ext_list(value, rule_obj, path):
    log.debug("value: %s", value)
    log.debug("rule_obj: %s", rule_obj)
    log.debug("path: %s", path)

    # Either raise some exception that you have defined your self
    # raise AssertionError('Custom assertion error in jinja_function()')

    # Or you should return True/False that will tell if it validated
    return True


def ext_map(value, rule_obj, path):
    log.debug("value: %s", value)
    log.debug("rule_obj: %s", rule_obj)
    log.debug("path: %s", path)

    # Either raise some exception that you have defined your self
    # raise AssertionError('Custom assertion error in jinja_function()')

    # Or you should return True/False that will tell if it validated
    return True
```

CHAPTER 7

The Community Guide

7.1 Testing

Install test/dev requirements with

```
pip install -r dev-requirements.txt
```

Run tests with

```
py.test
```

or if you want to test against all python versions and pep8

```
tox
```

7.2 Upgrading instructions

This document describes all major changes to the validation rules and the API that could cause existing schemas to break. If new types were added, they will not be described here because it will not break existing schemas.

7.2.1 1.5.x –> 1.6.0

ruamel.yaml is now possible to use as a drop-in replacement for PyYaml. Install it with *pip install 'pykwalify[ruamel]'* for production use and with *pip install -e '.[ruamel]'* for development use.

Several new keywords and types was added. It should bring more copmatiblity with original kwalify spec, but they can also break existsing schemas.

SECURITY: Please note that if you are executing user provided schemas there is a security risk in using the assert keyword.

Fixed several small bugs that have a high risk of causing validations to change behaviour from earlier versions. Many of the bugs was only found in complex schemas and data structures.

Default rule is now possible to be specefied with key = so if you have a key in your schema it will now be considered default validation and not a plain key.

New CLI flags was added. They are all optional and only introduce new (opt-in) functionality.

Reworked how UTF-8 is handled. Code should now be fully compatible internally with UTF-8/unicode encodings. Docs has been updated to help if you still have errors.

If the type check fails it will no longer continue to check any other keywords and report the errors for them. Before when it continued to process keywords, it would lead to errors that made no sense when keywords was not supposed to even be available for some types. This can cause less errors to appear when running your schemas.

A major change was done to the default type. It is now string so if you do not specify the type in the schema it will default back to string. This change is based on the original kwalify spec/code.

Keywords `desc`, `example`, `name` now enforces correct value type (str) even if the values them self have no impact on the validation.

7.2.2 1.4.x –> 1.5.0

Regex received some fixes, so make sure your schema files are still compatible and do not produce any new errors.

7.2.3 1.3.0 –> 1.4.0

Python 3.2 support has been dropped. It was going to be dropped when python 3.5 was released, but this made supporting python 2 & 3 at the same time easier now when fixing unicode support.

All logging and exception messages have been fixed to work with unicode characters in schema and data files. If you use this in lib mode then you should test your code to ensure it is still compatible.

If you use `RuleError` in your code, you must update to use the new `msg` and `error_key` variables.

If you use `SchemaConflict` in your code, you must update to use the new `msg` and `error_key` variables.

7.2.4 1.2.0 –> 1.3.0

Almost all validation error messages have been updated. If you are dependent on the error messages that is located in the variable `c.validation_errors` you must check if your code must be updated to use the new error messages.

When parsing the error messages yourself, you now have access to the exceptions and more detailed variables containing the `msg`, `path`, `key`, `regex` and `value` for each validation error.

7.2.5 1.1.0 –> 1.2.0

Because of the new multiple sequence item feature all old schemas should be tested to verify that they still work as expected and no regressions have been introduced.

7.2.6 Anything prior to 1.0.1 –> 1.1.0

In release 1.1.0 the type `any` was changed so that it now accept anything no matter what the value is. In previous releases it was only valid if the data was any of the implemented types. The one time your schema will break is if you use `any` and only want one of the implemented types.

7.3 Release Notes

7.3.1 1.6.0 (Jan 22, 2017)

New keywords:

- Add support for keyword *example*. It does nothing and have no validation done on it.
- Add support for keyword *version*. It does nothing and have no validation done on it.
- Add support for keyword *date* and added support keyword *format*. This can be used to validate many different types of *datetime* objects.
- Add support for keyword *length*. It is very similar to *range* but works primarily string types.
- Add support for keyword *assert*. It works by running the python code *assert <assert-expr>* and check if any exception is raised. This feature is considered dangerouns becuase there is only simple logic to prevent escaping out from validation.

Bug fixes:

- Fixed a bug where regexes marked as 'required' in a map were matched as strings, rather than regexes.
- Fixed a bug where the type validation did not work when schema specefied a sequence of map objects. It now outputs a proper *...is not a dict...* error instead.
- Fixed a bug in *unique* validation when a key that it tried to lookup in the data would not exists. Now it just ignores that it did not find any value becuase a missing value do not impact validation.
- Fixed a bug with keyword *ident* when the rule value was verified to be a *boolean*. It now only accepts *boolean* values as expected.
- Fixed a bug where if *allowempty* was specefied in a mapping type inside a sequence type then it would not properly validate.
- Fixed a bug where loaded extensions would not allways work in complex & nested objects.
- Fixed a major bug in very nested *seq* schemas where if the schema expected another *seq* but the value was something else it would not raise it as a validation error. This has now been fixed and now raises the proper error.
- Fixed a bug where include directive would not work in all cases when used inside a key in a mapping block.

New features:

- It is now possible to specify a default rule when using a mapping. The rule will be used whenever no other key could be found. This is a port of a missing feature from original kwalify implementation.
- Added new helper method *keywords* to *Rule* class that can output all active keywords for any *Rule* object. This helps when debugging code to be able to easily dump what all active keywords for any *Rule* object.
- Added new cli flag *–strict-rule-validation* that will validate that all used keywords in all *Rule* objects only uses the rules that is supported by each defined type. If you only use a *Core* object then set *strict_rule_validation=True* when creating the *Core* object instance. This feature is opt-in in this releaes but will be mandatory in *releases >= 1.7.0*.
- Added new cli flag *–fix-ruby-style-regex* that will trim slashes from ruby style regex/patterns. When using this flag the first and last / will be trimmed of the pattern before running validation. If you only use a *Core* object then set *fix_ruby_style_regex=True* when creating the *Core* object instance. Default behaviour will still be that you should use python style regex values but this flag can help in some cases when you can't change the schema.
- Added new cli flag *–allow-assertions* that will enable the otherwise blocked keyword *assert*. This flag was introduced so that pykwalify would not assert assertions without user controll. Default behaviour will be to raise a *CoreError* is assertion is used but not allowed explicitly. If you only use a *Core* object then set *allow_assertions=True* when creating the *Core* object instance.

Changed behaviour:

- Removed the force of *UTF-8* encoding when importing pykwalify package. It caused issues with *jyp-iter notebooks* on python 2.7.x Added documentation in Readme regarding the suggested solution to use *PYTHONIOENCODING=UTF-8* if the default solution do not work.
- Validation do no longer continue to process things like *pattern*, *regex*, *timestamp*, *range* and other additional checks if the type check fails. This can cause problems where previous errors will now initially be silenced when the typecheck for that value fails, but reappear again when the type check is fixed. (sbrunner)
- Catches *TypeError* when doing regex validation. That happens when the value is not a parsable string type.
- Checking that the value is a valid dict object is now done even if the mapping keyword is not specefied in the schema. This makes that check more eager and errors can apear that previously was not there.
- Changed the sane default type if that key is not defined to be *str*. Before this, type had to be defined every time and the default type did not work as expected. This is a major change and can cause validation to either fail or to stop failing depending on the case.
- Changed validation for if a value is required and a value in a list for example is *None*. It now adds a normal validation errors instead of raising a *CoreError*.
- Value for keyword *desc* now *MUST* be a string or a *RuleError* will be raised.
- Value for keyword *example* now *MUST* be a string or a *RuleError* will be raised.
- Value for keyword *name* now *MUST* be a string or a *RuleError* will be raised.

General changes:

- Ported alot of testcases directly from *Kwalify* test data (*test-validator.yaml -> 30f.yaml & 43s.yaml*) so that this lib can have greater confidence that rules is implemented in the same way as *Kwalify*.
- Refactored *test_core_files* method to now accept test files with multiple of documents. The method now tries to read all documents from each test file and run each document seperatly. It now alos reports more detailed about what file and document that fails the test to make it easier to track down problems.
- Major refactoring of test files to now be grouped based on what they are testing instead of a increased counter that do not represent anything. It will be easier to find out what keywords lack tests and what keywords that have enough tests.

7.3.2 1.5.2 (Nov 12, 2016)

- Convert all documentation to readthedocs
- True/False is no longer considered valid integer
- python3 'bytes' objects is now a valid strings and text type
- The regular PyYaml support is now deprecated in favor of ruamel.yaml, see the following link for more details about PyYaml being deprecated https://bitbucket.org/xi/pyyaml/issues/59/has-this-project-been-abandoned PyYaml will still be possible to use in the next major release version (1.6.0) but removed in release (1.7.0) and forward.
- ruamel.yaml is now possible to install with the following command for local development *pip install -e '.[ruamel]'* and for production, use *pip install 'pykwalify[ruamel]'*
- ruamel.yaml is now used before PyYaml if installed on your system
- Fixed a bug where scalar type was not validated correctly.
- Unpin all dependencies but still maintain a minimum versions of each lib
- Allowed mixing of regex and normal keywords when matching a string (jmacarthur)

7.3.3 1.5.1 (Mar 6, 2016)

- Improvements to documentation (scottclowe).
- Improved code linting by reworking private variables in Rule class to now be properties and updated all code that used the old way.
- Improved code linting by reworking all Log messages to render according to pep standard. (By using %s and passing in variables as positional arguments)
- Fix bug when validating sequence and value should only be unicode escaped when a string
- Improve validation of timestamps.
- Improve float validation to now accept strings that is valid ints that uses scientific notation, "1e-06" for example.
- Update travis to test against python 3.6

7.3.4 1.5.0 (Sep 30, 2015)

- float / number type now support range restrictions
- ranges on non number types (e.g. seq, string) now need to be non negative.
- Fixed encoding bug triggered when both regex matching-rule 'any' and 'all' found keyword that failed regex match. Added failure unit tests to cover regex matching-rule 'any' and 'all' during failed regex match. Updated allowed rule list to include matching-rule 'all'.
- Changed _validate_mappings method from using re.match to re.search. This fixes bug related to multiple keyword regex using matching-rule 'any'. Added success unit tests to test default, 'any', and 'all' matching-rule.

7.3.5 1.4.1 (Aug 27, 2015)

- Added tests to sdist to enable downstream packaging to run tests. No code changes in this release.

7.3.6 1.4.0 (Aug 4, 2015)

- Dropped support for python 3.2 becuase of unicode literals do not exists in python 3.2.
- Fixed logging & raised exceptions when using unicode characters inside schemas/data/filenames.
- Reworked all RuleError exceptions to now have better exception messages.
- RuleError exceptions now have a unique 'error_key' that can make it easier to identify what error it is.
- Paths for RuleErrors have been moved inside the exception as a variable.
- Rewrote all SchemaConflict exceptions to be more human readable.

7.3.7 1.3.0 (Jul 14, 2015)

- Rewrote most of the error messages to be more human readable. See *docs/Upgrade Instructions.md* for more details.
- It is now possible to use the exceptions that was raised for each validation error. It can be found in the variable *c.validation_errors_exceptions*. They contain more detailed information about the error.

7.3.8 1.2.0 (May 19, 2015)

- This feature is NEW and EXPERIMENTAL. Implemented support for multiple values inside in a sequence. This will allow the defenition of different types that one sequence can contain. You can either require each value in the sequence to be valid against one to all of the different possibilities. Tests show that it still maintains backward compatibility with all old schemas but it can't be guarantee. If you find a regression in this release please file a bug report so it can be fixed ASAP.
- This feature is NEW and EXPERIMENTAL. Added ability to define python files that can be used to have custom python code/functions that can be called on all types so that custom/extra validation can be done on all data structures.
- Add new keyword 'func' that is a string and is used to point to a function loaded via the extension system.
- Add new keyword 'extensions' that can only be used on the top level of the schema. It is should be a list with strings of files that should be loaded by the extension system. Paths can be relative or absolute.
- New cli option '-e FILE' or '–extension FILE' that can be used to load extension files from cli.
- Fixed a bug where types did not raise exceptions properly. If schema said it should be a map but data was a sequence, no validation error was raised in earlier versions but now it raises a 'NotSequenceError' or 'NotMappingError'.

7.3.9 1.1.0 (Apr 4, 2015)

- Rework cli string that docopt uses. Removed redundant flags that docopt provides [–version & –help]
- Add support for timestamp validation
- Add new runtime dependency 'python-dateutil' that is used to validate timestamps
- Change how 'any' keyword is implemented to now accept anything and not just the implemented types. (See Upgrade Instructions document for migration details)

7.3.10 1.0.1 (Mar 8, 2015)

Switched back to semantic version numbering for this lib.

- After the release of *15.01* the version schema was changed back from the <year>.<month> style version schema back to semantic version names. One big problem with this change is that *pypi* can't handle the change back to semantic names very well and because of this I had to remove the old releases from pypi and replace it with a single version *1.0.1*.
- No matter what version you were using you should consider upgrading to *1.0.1*. The difference between the two versions is very small and contains mostly bugfixes and added improvements.
- The old releases can still be obtained from *github.com* and if you really need the old version you can add the download url to your *requirements.txt* file.

7.3.11 15.01 (Jan 17, 2015)

- Fixed a bug in unique validation for mapping keys [See: PR-12] (Gonditeniz)

7.3.12 14.12 (Dec 24, 2014)

- Fixed broken regex matching on map keys.
- Source files with file ending *.yml* can now be loaded
- **Added aliases to some directives to make it easier/faster to write**
  - *sequence –> seq*
  - *mapping –> map*
  - *required –> req*
  - *regex –> re*
- Reworked all testing files to reduce number of files

7.3.13 14.08 (Aug 24, 2014)

- First version to be uploaded to pypi
- Keyword 'range' can now be applied to map & seq types.
- Added many more test files.
- Keyword 'length' was removed because 'range' can handle all cases now.
- Keyword 'range' now correctly checks the internal keys to be integers
- Major update to testing and increased coverage.

7.3.14 14.06.1 (Jun 24, 2014)

- New feature "partial schema". Define a small schema with a ID that can be reused at other places in the schema. See readme for details.
- New directive "include" that is used to include a partial schema at the specefied location.
- Cli and Core() now can handle multiple schema files.
- Directive "pattern" can no longer be used with map to validate all keys against that regex. Use "regex;" inside "mapping:"
- 'none' can now be used as a type
- Many more tests added

7.3.15 14.06 (Jun 7, 2014)

- New version scheme [YY.MM(.Minor-Release)]
- Added TravisCI support
- Update runtime dependency docopt to 0.6.1
- Update runtime dependency pyyaml to 3.11
- Huge refactoring of logging and how it works. Logging config files is now removed and everything is alot simpler
- Cleanup some checks that docopt now handles

- New keyword "regex;<regex-pattern>" that can be used as a key in map to give more flexibility when validating map keys
- New keyword "matching-rule" that can be used to control how keys should be matched
- Added python 3.4 & python 2.7 support (See TravisCI tests for status)
- Dropped python 3.1 support
- Alot of refactoring of testing code.
- Tests should now be runned with "nosetests" and not "python runtests.py"
- Refactored alot of exceptions to be more specific (SchemaError and RuleError for example) and not a generic Exception
- Parsed rules is now stored correctly in Core() so it can be tested from the outside

7.3.16 0.1.2 (Jan 26, 2013)

- Added new and experimental validation rule allowempty. (See README for more info)
- Added TODO tracking file.
- Reworked the CLI to now use docopt and removede argparse.
- Implemented more typechecks, float, number, text, any
- Now suports python 3.3.x
- No longer support any python 2.x.y version
- Enabled pattern for map rule. It enables the validation of all keys in that map. (See README for more info)
- Alot more test files and now tests source_data and schema_data input arguments to core.py
- Alot of cleanup in the test suit

7.3.17 0.1.1 (Jan 21, 2013)

- Reworked the structure of the project to be more clean and easy to find stuff.
- lib/ folder is now removed and all contents is placed in the root of the project
- All scripts is now moved to its own folder scripts/ (To use the script during dev the path to the root of the project must be in your python path somehow, recomended is to create a virtualenv and export the correct path when it activates)
- New make target 'cleanegg'
- Fixed path bugs in Makefile
- Fixed path bugs in Manifest

7.3.18 0.1.0 (Jan 20, 2013)

- Initial stable release of pyKwalify.
- All functions is not currently implemented but the cli/lib can be used but probably with some bugs.
- This should be considered a Alpha release used for bug and stable testing and to be based on further new feature requests for the next version.

- Implemented most validation rules from the original Java version of kwalify. Some is currently not implemented and can be found via [NYI] tag in output, doc & code.
- Installable via pip (Not the official online pip repo but from the releases folder found in this repo)
- Supports YAML & JSON files from cli and any dict/list data structure if used in lib mode.
- Uses pythons internal logging functionality and default logging output can be changed by changing logging.ini (python 3.1.x) or logging.yaml (python 3.2.x) to change the default logging output, or use -v cli input argument to change the logging level. If in lib mode it uses your implemented python std logging.

7.4 Authors

7.4.1 Code

- Grokzen (https://github.com/Grokzen)
- Markbaas (https://github.com/markbaas)
- Gonditeniz (https://github.com/gonditeniz)
- Comagnaw (https://github.com/comagnaw)
- Cogwirrel (https://github.com/cogwirrel)

7.4.2 Testing

- Glenn Schmottlach (https://github.com/gschmottlach-xse)

7.4.3 Documentation

- Grokzen (https://github.com/Grokzen)
- Scott Lowe (https://github.com/scottclowe)

7.5 Licensing