Apache Commons Validator Python

Sanjana Nandi, Juji Lau, Alicia Chu, Jessica Breuhaus

May 13, 2025

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

APACHE_COMMONS_VALIDATOR_PYTHON

1.1 apache_commons_validator_python package

1.1.1 Subpackages

apache_commons_validator_python.routines package

Subpackages

apache_commons_validator_python.routines.checkdigit package

Submodules

apache_commons_validator_python.routines.checkdigit.abstract_checkdigit module

Module Name: checkdigit.py

Description: Translates apache.commons.validator.routines.checkdigit.CheckDigit.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/AbstractCheckDigit.java

Author: Juji Lau

License (Taken from apache.commons.validator.routines.checkdigit.AbstractCheckDigit.java):

Changes:

Added serializeable and clone as abstract attributes

class `apache_commons_validator_python.routines.checkdigit.abstract_checkdigit.`**`AbstractCheckDigit`**

Bases: `CheckDigit`

Abstract class for CheckDigit interface.

apache_commons_validator_python.routines.checkdigit.checkdigit module

Module Name: checkdigit.py Description: Translates apache.commons.validator.routines.checkdigit.CheckDigit.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/CheckDigit.java Paraphrased from apache.commons.validator.routines.checkdigit.CheckDigit.java:

> Check Digit calculation and validation.
>
> The logic for validating check digits was previously embedded within specific code validation, which included format and length verification. The *CheckDigit* class separates the check digit calculation logic, making it easier to test and reuse.
>
> Although Commons Validator primarily focuses on validation, *CheckDigit* also defines behavior for calculating and generating check digits. This allows users to reuse the same logic for both validation and generation. For example, validator.routines.ISBNValidator makes specific use of this feature by providing the facility to validate ISBN-10 codes and then converting them to the new ISBN-13 standard.
>
> *CheckDigit* is used by the generic *CodeValidator* implementation.
>
> **Implementations:**
> See the package summary for a full list of implementations provided within Commons Validator.

Author: Juji Lau License (Taken from apache.commons.validator.routines.checkdigit.CheckDigit.java):

Changes:
Added serializeable and clone as abstract attributes

`class apache_commons_validator_python.routines.checkdigit.checkdigit.`**`CheckDigit`**

Bases: `ABC`

Check Digit calculation and validation.

The logic for validating check digits was previously embedded within specific code validation, which included format and length verification. The *CheckDigit* class separates the check digit calculation logic, making it easier to test and reuse.

Although Commons Validator primarily focuses on validation, *CheckDigit* also defines behavior for calculating and generating check digits. This allows users to reuse the same logic for both validation and generation. For example, validator.routines.ISBNValidator makes specific use of this feature by providing the facility to validate ISBN-10 codes and then converting them to the new ISBN-13 standard.

CheckDigit is used by the generic *CodeValidator* implementation.

`serializable`

Indicates if the object is serializable.

Type
bool

`clone`

Indicates if the object can be cloned.

Type
bool

`abstract calculate`(*code: str*) → str | CheckDigitException | None

Calculates the Check Digit for a code.

Parameters
code (*str*) – The code to calculate the Check Digit for. It must not include the checkdigit

Returns
The calculated Check Digit

Raises
CheckDigitException if an error occurs. –

`abstract property clone: bool`

`abstract is_valid`(*code: str*) → bool

Validates the check digit for the code.

Parameters
code (*str*) – The code to validate, the string must include the check digit.

Returns
True if the check digit is valid; False otherwise

`abstract property serializable: bool`

apache_commons_validator_python.routines.checkdigit.checkdigit_exception module

Module Name: checkdigit_exception.py Description: Translates apache.commons.validator.routines.checkdigit.CheckDigitException.jav
Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/CheckDigitException.java Parapphrased from apache.commons.validator.routines.checkdigit.CheckDigitException.java:

Check Digit calculation/validation error.

Author: Juji Lau License (Taken from apache.commons.validator.routines.checkdigit.CheckDigitException.java):

Changes:
Added self.__cause__ to allow propogation of Python's base Exceptions.

exception `apache_commons_validator_python.routines.checkdigit.checkdigit_exception.`**`CheckDigitException`**(

Bases: `Exception`

Exception raised for errors in check digit calculation or validation.

`message`

Explanation of the error.

Type
str

`cause`

Underlying exception that caused this error.

Type
Exception, optional

`serializable`

Indicates if the object is serializable.

Type
bool

`clone`

Indicates if the object can be cloned.

Type
bool

`clone = False`

`serializable = True`

apache_commons_validator_python.routines.checkdigit.ean13_checkdigit module

Module Name: ean13.py Description: Translates apache.commons.validator.routines.checkdigit.EAN13CheckDigit.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/EAN13CheckDigit.java Parapphrased from apache.commons.validator.routines.checkdigit.EAN13CheckDigit.java:

> Modulus 10 EAN-13 UPC ISBN-13 Check Digit calculation/validation. Check digit calculation is based on modulus 10 with digits in an odd position (from right to left) being weighted 1 and even position digits being weighted 3.
>
> **For further information see:**
>
> - EAN-13: https://en.wikipedia.org/wiki/European_Article_Number (Wikipedia - European Article Number)

- UPC: https://en.wikipedia.org/wiki/Universal_Product_Code (Wikipedia - Universal Product Code)
- ISBN-13: https://en.wikipedia.org/wiki/ISBN (Wikipedia - International Standard Book Number (ISBN))

Author: Juji Lau License (Taken from apache.commons.validator.routines.checkdigit.EAN13CheckDigit.java):

Changes:

`class apache_commons_validator_python.routines.checkdigit.ean13_checkdigit.`**`EAN13CheckDigit`**

Bases: `ModulusCheckDigit`

Modulus 10 EAN-13 / UPC / ISBN-13 Check Digit calculation and validation.

This class implements a check digit routine for the EAN-13 format, which is widely used for barcodes. The calculation follows the Modulus 10 algorithm, assigning different weights to digits based on their position.

`serializable`

Inherited from ModulusCheckDigit (True)

Type
bool

`clone`

Inherited from ModulusCheckDigit (False)

Type
bool

Constants:
EAN13_CHECK_DIGIT (EAN13CheckDigit): Singleton instance of this class. POSITION_WEIGHT (list[int]): Weighting given to digits depending on their right position

`class property` **`EAN13_CHECK_DIGIT`**

Enforces singleton behavior and returns the singleton instance of this validator.

Returns
A singleton instance of the validator.

apache_commons_validator_python.routines.checkdigit.isbn10_checkdigit module

Module Name: isbn10_checkdigit.py Description: Translates apache.commons.validator.routines.checkdigit.ISBN10CheckDigit.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/ISBN10CheckDigit.java Parapphrased from apache.commons.validator.routines.checkdigit.ISBN10CheckDigit.java:

> Modulus 11 ISBN-10 Check Digit calculation/validation. ISBN-10 Numbers are a numeric code except for the last (check) digit which can have a value of "X".
>
> Check digit calculation is based on modulus 11 with digits being weighted based by their position, from right to left with the first digit being weighted 1, the second 2 and so on. If the check digit is calculated as "10" it is converted to "X".
>
> **NOTE:** From 1st January 2007 the book industry will start to use a new 13 digit ISBN number (rather than this 10 digit ISBN number) which uses the EAN-13 / UPC standard (see EAN13CheckDigit).
>
> **For further information see:**
>
> - *Wikipedia - ISBN <https://en.wikipedia.org/wiki/ISBN>*
> - `ISBN-13 Transition Details <http://www.isbn.org/standards/home/isbn/transition.asp>

Author: Juji Lau License (Taken from apache.commons.validator.routines.checkdigit.ISBN10CheckDigit.java):

Changes:

`class apache_commons_validator_python.routines.checkdigit.isbn10_checkdigit.`**`ISBN10CheckDigit`**

Bases: `ModulusCheckDigit`

This class perfroms Modulus 11 ISBN-10 Check Digit calculation/validation. ISBN-10 Numbers are a numeric code except for the last (check) digit which can have a value of "X".

Check digit calculation is based on modulus 11 with digits being weighted based by their position, from right to left with the first digit being weighted 1, the second 2 and so on. If the check digit is calculated as "10" it is converted to "X".

`serializable`

Inherited from ModulusCheckDigit (True)

Type
bool

`clone`

Inherited from ModulusCheckDigit (False)

Type
bool

Constants:
ISBN10_CHECK_DIGIT (ISBN10CheckDigit): Singleton instance of this class.

`class property ISBN10_CHECK_DIGIT`

Enforces singleton behavior and returns the singleton instance of this validator.

Returns
A singleton instance of the validator.

apache_commons_validator_python.routines.checkdigit.isin_checkdigit module

Module Name: isin_checkdigit.py Description: Translates apache.commons.validator.routines.checkdigit.ISINCheckDigit.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/ISINCheckDigit.java

Author: Alicia Chu License (Taken from apache.commons.validator.routines.checkdigit.ISINCheckDigit.java):

Changes: - StringBuilder -> list of strings

`class apache_commons_validator_python.routines.checkdigit.isin_checkdigit.ISINCheckDigit`

Bases: `ModulusCheckDigit`

Modulus 10 ISIN (International Securities Identifying Number) Check Digit calculation and validation.

ISINs are 12-character alphanumeric identifiers used for securities. This validator uses the Modulus 10 Double Add Double method: - Every second digit is weighted by 2, starting from the right. - Alphabetic characters are converted to values: A=10, B=11, …, Z=35. - Weighted digits over 9 are split and summed (e.g., 18 -> 1 + 8 = 9).

apache_commons_validator_python.routines.checkdigit.luhn_checkdigit module

Module Name: luhn_checkdigit.py Description: Translates apache.commons.validator.routines.checkdigit.LuhnCheckDigit.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/LuhnCheckDigit.java

Author: Alicia Chu License (Taken from apache.commons.validator.routines.checkdigit.LuhnCheckDigit.java):

Changes:

`class apache_commons_validator_python.routines.checkdigit.luhn_checkdigit.`**`LuhnCheckDigit`**

Bases: `ModulusCheckDigit`

This class is used to validate check digits using the Luhn (modulus 10) algorithm, which is commonly applied to credit card numbers and other identification numbers.

The Luhn algorithm weights digits from right to left, doubling every second digit and subtracting 9 if the result exceeds 9. This helps detect common data entry errors such as transpositions and single-digit mistakes.

`LUHN_CHECK_DIGIT`

Singleton instance of this class.

Type

LuhnCheckDigit

`LUHN_CHECK_DIGIT = <apache_commons_validator_python.routines.checkdigit.luhn_checkdigit.LuhnCheckDigit object>`

apache_commons_validator_python.routines.checkdigit.modulus_checkdigit module

Module Name: modulus_checkdigit.py Description: Translates apache.commons.validator.routines.checkdigit.ModulusCheckDigit.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/checkdigit/ModulusCheckDigit.java Parapphrased from apache.commons.validator.routines.checkdigit.ModulusCheckDigit.java:

Abstract Modulus Check digit calculation/validation:

Provides a base class for building Modulus Check Digit routines. This implementation only handles single-digit numeric codes, such as EAN-13. For alphanumeric codes such as EAN-128 you will need to implement/override the *toInt()* and *toChar()* methods.

Author: Juji Lau License (Taken from apache.commons.validator.routines.checkdigit.ModulusCheckDigit.java):

Changes:

In ModulusCheckDigit._to_int() for the argument, ***final char character*****:**

- I accept a Python str of len(1) in the place of Java’s char.
- Python does not support single characters
- I raise a ValueError exception via CheckDigitException if the input is not a string of length 1.

`class apache_commons_validator_python.routines.checkdigit.modulus_checkdigit.`**`ModulusCheckDigit`**`(*, modulus: int = 10)`

Bases: `AbstractCheckDigit`

Abstract base class for Modulus Check Digit calculation and validation.

This class provides a foundation for implementing modulus-based check digit routines. It supports single-digit numeric codes like EAN-13. For alphanumeric codes (e.g., EAN-128) override the *_to_int()* and *_to_char()* methods.

`MODULUS_10`

Type
int

`MODULUS_11`

Type
int

`modulus`

Type
int

`serializable`

Indicates if the object is serializable.

Type
bool

`clone`

Indicates if the object can be cloned.

Type
bool

`MODULUS_10: Final[int] = 10`

`MODULUS_11: Final[int] = 11`

`calculate``(code: str)` → str | CheckDigitException | None

Calculate a modulus heck Digit for a code which does not yet have one.

Parameters
code (*str*) – The code for which to calculate the Check Digit; the check digit should not be included

Returns
The calculated Check Digit

Raises
CheckDigitException if an error occurs calculating the check digit. –

`clone = False`

`is_valid(code: str) → bool`

Validate a modulus check digit for a code.

Parameters
`code` (*str*) – The code to validate.

Returns
True if the check digit is valid. False otherwise.

`property modulus: int`

Gets the modulus value this check digit routine is based on.

`serializable = True`

`static sum_digits(number: int) → int`

Add together the individual digits in a number.

Parameters
`number` (*int*) – The number whose digits are to be added

Returns
The sum of the digits.

Module contents

Submodules

apache_commons_validator_python.routines.abstract_calendar_validator module

Module Name: abstract_calendar_validator.py

Description: Translates apache.commons.validator.routines.AbstractCalendarValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/AbstractCalendarValidator.java

Author: Juji Lau

License (Taken from apache.commons.validator.routines.AbstractCalendarValidator.java):

Changes:

- Removed Java's Calendar fields that don't have an equivalent datetime field.

Modified `AbstractCalendarValidator.compare(Calendar value, Calendar compare, int field)` have users pass in a string instead of an int. Python's datetime validator does not map its properties to integers, hence using integers will be confusing to Python users.

Implemented `parse()` in the concrete subclasses instead.

`class apache_commons_validator_python.routines.abstract_calendar_validator.AbstractCalendarValidator`(*str*
boo
dat
int
tim
int

Bases: `AbstractFormatValidator`

Abstract class for Date/Time/Calendar validation.

This is a base class for building Date/Time Validators using format parsing.

`date_style`
: The date style to use for Locale validation.

Type
: int

`time_style`
: The time style to use for Locale validation.

Type
: int

`int2str_style`
: Maps the integer date and time style to a string argument for `babel.format()`.

Type
: dict[int, str]

`serializable`
: Indicates if the object is serializable.

Type
: bool

`cloneable`
: Indicates if the object can be cloned.

Type
: bool

`cloneable = False`

`format`(*, *value: object = None*, *pattern: str = None*, *locale: str | Locale = None*, *time_zone: timezone = None*) → str

Format an object into a string using the specified pattern and/or locale.

Parameters

- **value** (*object*) – The value validation is being performed on.
- **pattern** (*str*) – The pattern used to format the value.
- **locale** (*str*) – A locale string (e.g., "en_US") used for formatting. If locale is `None` or empty, the system default is used.
- **time_zone** (*timezone*) – The timezone used to format the date, If `None`, the system default will be used unless value is a *datetime*.

Returns
: The value formatted as a String.

`is_valid`(***, *value: str*, *pattern: str | None = None*, *locale: str | None = None*) → bool

Validate using the specified locale.

Parameters

- **value** (*str*) – The value validation is being performed on.
- **pattern** (*str*) – The pattern used to format the value.
- **locale** (*str*) – The locale to use for the Format, defaults to the system if None.

Returns

`True` if the value is valid; `False` otherwise.

serializable = True

apache_commons_validator_python.routines.abstract_format_validator module

Module Name: abstract_format_validator.py

Description: Translates apache.commons.validator.routines.AbstractFormatValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/AbstractFormatValidator.java

Author: Jessica Breuhaus

class `apache_commons_validator_python.routines.abstract_format_validator.`**AbstractFormatValidator**(*strict: bool*)

Bases: `ABC`

Abstract class for format based Validation.

This is a base class for building Date and Number Validators using format parsing.

cloneable = True

format(*value*, *pattern: str = None*, *locale: str = None*)

Format an object into a string using the specified pattern or locale.

Parameters

- **value** (*int or float*) – The value to format into a string.
- **pattern** (*str*) – The (optional) string format to use to format the string.
- **locale** (*str*) – The (optional) locale to use to format the string.

Returns

The value formatted as a str.

is_valid(*value: str*, *pattern: str = None*, *locale: str = None*)

Validate using the specified pattern and/or locale or the default if no pattern and/or locale is given.

Parameters

- **value** (*str*) – The value validation is being performed on.
- **pattern** (*str*) – The (optional) regex pattern used to validate the value against.
- **locale** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns

True if the value is valid.

serializable = True

property strict

Indicates whether validated values should adhere strictly to the format used.

Returns

True if strict parsing will be used.

apache_commons_validator_python.routines.abstract_number_validator module

Module Name: abstract_number_validator.py

Description: Translates apache.commons.validator.routines.AbstractNumberValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/AbstractNumberValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.AbstractNumberValidator.java):

class apache_commons_validator_python.routines.abstract_number_validator.**AbstractNumberValidator**(*strict: bool*, *format_typ int*, *allow_frac bool*)

Bases: *AbstractFormatValidator*

Abstract base class for Number Validation.

This is a base class for building Number Validators.

Once a value has been successfully converted the following methods can be used to perform minimum, maximum and range checks:

- min_value() checks whether the value is greater than or equal to a specified minimum.
- max_value() checks whether the value is less than or equal to a specified maximum.
- is_in_range() checks whether the value is within a specified range of values.

`STANDARD_FORMAT`

Used to indicate the standard format.

Type
int

`CURRENCY_FORMAT`

Used to indicate the currency format.

Type
int

`PERCENT_FORMAT`

Used to indicate the percent format.

Type
int

`CURRENCY_FORMAT: Final[int] = 1`

`PERCENT_FORMAT: Final[int] = 2`

`STANDARD_FORMAT: Final[int] = 0`

`property allow_fractions`

Indicates whether the number being validated is a decimal or integer.

Returns
True if decimals are allowed or *False* if ints only.

`format`(*value*, *pattern: str = None*, *locale: str = None*)

Format an object into a string using the specified pattern or locale.

Parameters

- **value** (*int or float*) – The value to format into a string.
- **pattern** (*str*) – The (optional) string format to use to format the string.
- **locale** (*str*) – The (optional) locale to use to format the string.

Returns
The value formatted as a str.

`property format_type`

Indicates the type of format created by this validator instance. The three types are STANDARD_FORMAT, CURRENCY_FORMAT, and PERCENT_FORMAT.

Returns
The format type created.

`is_in_range`(*value*, *min_val*, *max_val*)

Check if the value is within a specified range.

Parameters

- **value** (*int or float*) – The value to check.

- **min_val** (*int or float*) – The minimum value of the range.
- **max_val** (*int or float*) – The maximum value of the range.

Returns
True if the value is within the specified range.

is_valid(*value: str*, *pattern: str = None*, *locale: str = None*)
Validate using the specified locale.

Parameters

- **value** (*str*) – The value validation is being performed on.
- **pattern** (*str*) – The (optional) regex pattern used to validate the value against.
- **locale** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns
True if the value is valid.

max_value(*value*, *max_val*)
Check if the value is less than or equal to a maximum.

Parameters

- **value** (*int or float*) – The value to check.
- **max_value** (*int or float*) – The maximum value.

Returns
True if the value is less than or equal to the maximum.

min_value(*value*, *min_val*)
Check if the value is greater than or equal to a minimum.

Parameters

- **value** (*int or float*) – The value to check.
- **min_value** (*int or float*) – The minimum value.

Returns
True if the value is greater than or equal to the minimum.

apache_commons_validator_python.routines.big_decimal_validator module

Module Name: big_decimal_validator.py

Description: Translates apache.commons.validator.routines.BigDecimalValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/BigDecimalValidator.java

Author: Jessica Breuhaus

class apache_commons_validator_python.routines.big_decimal_validator.**BigDecimalValidator**(*strict: bool = True*, *format_type: int = 0*, *allow_fractions: bool = True*)

Bases: *AbstractNumberValidator*

BigDecimal Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to a big decimal to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted big decimal value.

Fraction/decimal values are automatically rounded to the appropriate length.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

classmethod get_instance()

Gets the singleton instance of this validator.

Returns

A singleton instance of the validator.

validate(*value: str*, *pattern: str = None*, *locale=None*)

Validate/convert a big decimal using the optional pattern and/or locale.

Parameters

- **value** (*str*) – The value validation is being performed on.

- **pattern** (*str*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- **locale** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns
The parsed big decimal (as a float) if valid or *None* if invalid.

apache_commons_validator_python.routines.big_integer_validator module

Module Name: big_integer_validator.py

Description: Translates apache.commons.validator.routines.BigIntegerValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/BigIntegerValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.BigIntegerValidator.java):

class apache_commons_validator_python.routines.big_integer_validator.**BigIntegerValidator**(*strict: bool = True*, *format_type: int = 0*)

Bases: *AbstractNumberValidator*

BigInteger Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to a big integer to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted big integer value.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.

- using the default format for a specified locale.
- using a specified pattern with a specified locale.

classmethod get_instance()

Gets the singleton instance of this validator.

Returns
A singleton instance of the validator.

validate(*value: str*, *pattern: str = None*, *locale=None*)

Validate/convert a big integer using the optional pattern and/or locale.

Parameters

- **value** (*str*) – The value validation is being performed on.
- **pattern** (*str*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- **locale** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns
The parsed big integer (as an int) if valid or *None* if invalid.

apache_commons_validator_python.routines.byte_validator module

Module Name: byte_validator.py

Description: Translates apache.commons.validator.routines.ByteValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/ByteValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.ByteValidator.java):

class apache_commons_validator_python.routines.byte_validator.**ByteValidator**(*strict: bool = True*, *format_type: int = 0*)

Bases: *AbstractNumberValidator*

Byte Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to a byte to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted byte value.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

`BYTE_MIN`

The value of the maximum allowed byte (-128).

Type
int

`BYTE_MAX`

The value of the minimum allowed byte (127).

Type
int

`BYTE_MAX: Final[int] = 127`

`BYTE_MIN: Final[int] = -128`

`classmethod get_instance()`

Gets the singleton instance of this validator.

Returns
A singleton instance of the validator.

`validate`(*value: str*, *pattern: str = None*, *locale=None*)

Validate/convert a byte using the optional pattern and/or locale.

Parameters

- **value** (*str*) – The value validation is being performed on.
- **pattern** (*str*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- **locale** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns
The parsed byte (as an int) if valid or *None* if invalid.

apache_commons_validator_python.routines.calendar_validator module

Module Name: calendar_validator.py

Description: Translates apache.commons.validator.routines.CalendarValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/CalendarValidator.java

This file is meant to translate Java's `Calendar` class. However, since Python's `datetime.datetime` class is much more closely functional to Java's `Calendar` class, this file will be validating Python's `datetime.datetime` class.

Author: Juji Lau

`class apache_commons_validator_python.routines.calendar_validator.`**`CalendarValidator`**`(*, `*strict: bool = True*, *date_style: int = 3*`)`

Bases: `AbstractCalendarValidator`

Calendar Validation and Conversion Routines (using Python datetime objects)

This module provides a collection of methods for validating and converting a string representing a date and/or time into a Python datetime (or date/time) object. Parsing can be performed using various mechanisms:

- Using the default format for the default locale.
- Using a specified pattern with the default locale.
- Using the default format for a specified locale.
- Using a specified pattern with a specified locale.

For each of the above mechanisms, validation function implementations are provided which either use the system's default timezone or allow a timezone to be explicitly specified.

Usage:

- Use one of the is_valid() methods to simply check if a given string is valid.
- Use one of the validate() methods to both validate the input and return a

converted datetime (or date/time) object.

Additionally, methods are provided to create datetime objects adjusted for different time zones if the system default is not appropriate. Alternatively, the `adjust_to_timezone()` method can be used to modify the timezone of an existing `datetime` object.

Once a value has been successfully converted, several comparison methods are available to perform date arithmetic and comparison checks:

- compare_dates(): Compares the day, month, and year of two datetime objects.
- compare_weeks(): Compares the week and year of two datetime objects.
- compare_months(): Compares the month and year of two datetime objects.
- **compare_quarters(): Compares the quarter (computed from the month) and year**
  of two datetime objects.

These compare methods return 0 if the arguments are equal, -1 if the first argument is earlier, or +1 if it is later.

To allow the same parsing mechanism used for input validation to be used for output formatting, corresponding format() methods are provided. These methods enable formatting of datetime objects by:

- Using a specified pattern.
- Using the format for a specified locale.
- Using the format for the default locale.

`VALDIATOR`

An instance of this validator.

Type
CalendarValidator

`serializable`

Indicates if the object is serializable (class attribute).

Type
bool

`cloneable`

Indicates if the object can be cloned (class attribute).

Type
bool

`classmethod adjust_to_time_zone`(*value: datetime*, *time_zone: tzinfo*) → datetime

Adjusts a datetime's value to a different time_zone.

Parameters

- **value** (*datetime*) – The value to adjust.
- **time_zone** (*tzinfo*) – The new time zone to use to adjust the Calendar to.

Returns
A new datetime with the values adjusted.

`cloneable = False`

`compare_dates`(*value: datetime*, *compare: datetime*) → int

Compare Dates (day, month and year - not time)

Parameters

- **value** (*datetime*) – The datetime value to check.
- **compare** (*datetime*) – The datetime to compare the value to.

Returns
0 if the dates are equal, -1 if the first date is less than the second. +1 if the first date is greater than the second.

`compare_months`(*value: datetime*, *compare: datetime*) → int

Compare Months (month and year).

Parameters

- **value** (*datetime*) – The datetime value to check.
- **compare** (*datetime*) – The datetime to compare the value to.

Returns
0 if the months are equal, -1 if the first month is less than the second. +1 if the first month is greater than the second.

`compare_quarters`(*value: datetime*, *compare: datetime*, *month_of_first_quarter: int = 1*) → int

Compare Quarters (quarter and year).

Parameters

- **`value`** (*`datetime`*) – The datetime value to check.
- **`compare`** (*`datetime`*) – The datetime to compare the value to.
- **`month_of_first_quarter`** (*`int`*) – The month the first quarter starts (default is 1 or Jaunuary)

Returns
0 if the quarters are equal, -1 if the first quarters is less than the second. +1 if the first quarters is greater than the second.

`compare_weeks`(*value: datetime*, *compare: datetime*) → int

Compare Weeks (week and year).

Parameters

- **`value`** (*`datetime`*) – The datetime value to check.
- **`compare`** (*`datetime`*) – The datetime to compare the value to.

Returns
0 if the weeks are equal, -1 if the first week is less than the second. +1 if the first week is greater than the second.

`compare_years`(*value: datetime*, *compare: datetime*) → int

Compare Years.

Parameters

- **`value`** (*`datetime`*) – The datetime value to check.
- **`compare`** (*`datetime`*) – The datetime to compare the value to.

Returns
0 if the years are equal, -1 if the first year is less than the second. +1 if the first year is greater than the second.

`classmethod get_instance()`

Returns the singleton instance of the CalendarValidator.

`serializable = True`

`validate`(*value: str = None*, *pattern: str = None*, *locale: str | None = None*, *time_zone: tzinfo | None = None*) → datetime

Validate/convert a *datetime* using the specified *locale* and *timezone*. If these arguments are not provided, the system default will be used.

Parameters

- **`value`** (*`str`*) – The value validation is being performed on.
- **`pattern`** (*`str`*)
- **`dt_locale`** (*`str`*) – A locale string (e.g., "en_US") used for the date formatting. If `None`, the default will be used.

- **time_zone** (*tzinfo/timezone*)

Returns
The parsed *datetime* if valid or `None` if invalid.

apache_commons_validator_python.routines.code_validator module

Module Name: code_validator.py Description: Translates apache.commons.validator.routines.CodeValidator.java

Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/CodeValidator.java Paraphrased from apache.commons.validator.routines.CodeValidator:

This class provides generic Code Validation, providing format, minimum/maximum length, and check digit validations.

This class performs the following validations on a code:

- If the code is `None`, return `None/False` as appropriate.
- Trim the input. If the resulting code is empty, return `None/False` as appropriate.
- Check the *format* of the code using a *regular expression* (if specified).
- Check the *minimum* and *maximum* length (if specified) of the *parsed* code (that is, parsed by the *regular expression*).
- Perform `CheckDigit` validation on the parsed code (if specified).
- The `validate()` method returns the trimmed, parsed input (or `None` if validation failed).

Note: The `is_valid()` method will return `True` if the input passes validation. Since this includes trimming as well as potentially dropping parts of the input, it is possible for a string to pass validation but fail the check digit test if passed directly to it. Check digit routines generally don't trim input nor do they check the format/length.

To ensure valid input is passed to a method, use `validate()` as follows:

```python
valid = validator.validate(input)
if valid is not None:
    some_method(valid)
```

The validator should be configured with the appropriate regular expression, minimum/maximum length, and check digit validator before calling one of the two validation methods:

- `is_valid()`
- `validate()`

Codes often include *format* characters—such as hyphens—to improve human readability. These can be removed prior to length and check digit validation by specifying them as a *non-capturing* group in the regular expression (i.e., using the `(?: )` notation).

Alternatively, avoid using parentheses except for the parts you want to capture.

since
1.4

Author: Juji Lau License (Taken from apache.commons.validator.routines.ISBNValidator):

Changes:

- Removed getInstance() which supports singletone behavior for a Java class. In here, singleton behavior is implicit.
- Added a setter for self.convert. getInstance(convert) provided a way to do this, but now that it's removed, this adds a new way.

`class apache_commons_validator_python.routines.code_validator.CodeValidator`(*, *regex: str = None*, *regex_validator: RegexValidator = None*, *length: int = None*, *min_length: int = -1*, *max_length: int = -1*, *checkdigit: CheckDigit = None*)

Bases: `object`

This class performs generic code validation, including format checks using regular expressions, minimum/maximum length validation, and check digit validation.

It ensures that:

- If the input code is *None*, returns *None*.
- It trims the input code and checks if the result is non-empty.
- It validates the code format using a regular expression (if specified).
- It checks the minimum and maximum length of the code (if specified).
- It optionally performs check digit validation.

`regex_validator`

The regular expression validator for the code format.

Type

RegexValidator

`checkdigit`

The check digit validation routine.

Type

CheckDigit

`min_length`

The minimum length of the code.

Type
int

`max_length`

The maximum length of the code.

Type
int

`serializable`

Indicates if the object is serializable (class attribute).

Type
bool

`cloneable`

Indicates if the object can be cloned (class attribute).

Type
bool

`property checkdigit: CheckDigit`

Returns the checkdigit attribute.

`cloneable = False`

`is_valid`(*input: str*) → bool

Validates the input by calling validate(). returning either True or False.

Parameters
`input` (*str*) – The code to validate and check for validity.

Returns
False if the return value of validate() is None. *True* otherwise.

`property max_length: int`

Returns the max_length attribute.

`property min_length: int`

Returns the min_length attribute.

`property regex_validator: RegexValidator | None`

Returns the regex_validator attribute.

`serializable = True`

`validate`(*input: str*) → str | None

Validate the input returning either the valid input or None if the input is invalid Note: This method trims the input and if *self.regex_validator* is set, it may also

change the input as part of the validation.

Parameters
`input` (*str*) – The code to validate.

Returns
The validated input if the code is valid *None* if the code is invalid

apache_commons_validator_python.routines.credit_card_validator module

Module Name: credit_card_validator.py

Description: Translates apache.commons.validator.routines.CreditCardValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/CreditCardValidator.java

Author: Alicia Chu

License (Taken from apache.commons.validator.routines.CreditCardValidator.java):

Changes:

In Java, the CreditCardValidator class has multiple overloaded constructors but , in Python, all that flexibility is handled in a single __init__ method.

`class apache_commons_validator_python.routines.credit_card_validator.`**`CreditCardValidator`**(*options: int = 15, credit_card_valida list[CodeValidator | None = None, credit_card_range list[*CreditCardVal *| None = None*)

Bases: `object`

Validates credit card numbers based on known issuer patterns, numeric format, and Luhn check digit rules.

This class supports multiple built-in credit card types such as Visa, MasterCard, American Express, Discover, Diners, and VPay. It can also be configured with custom validation rules, either through regular expressions or prefix-based range validation.

`serializable`

Indicates that instances can be serialized (always True).

Type

bool

`cloneable`

Indicates whether instances can be cloned (always False).

Type
bool

Usage:

- Use the default constructor to validate common card types.
- Use *generic_credit_card_validator()* to validate any numeric card number within a length range using only Luhn check.
- Extend with custom *CodeValidator* or range-based validators for specialized card types.

`AMEX: Final[int] = 1`

`AMEX_VALIDATOR: Final = <src.apache_commons_validator_python.routines.code_validator.CodeValidator object>`

Diners Card Validator <ul> <li>300xxx - 305xxx (14)</li> <li>3095xx (14)</li> <li>36xxxx (14)</li> <li>38xxxx (14)</li> <li>39xxxx (14)</li> </ul>

`class CreditCardRange`(*low: str*, *high: str | None = None*, *min_len: int = -1*, *max_len: int = -1*, *lengths: list[int] | None = None*)

Bases: `object`

Represents a credit card number range for validating issuer prefix (IIN) and permissible card number lengths.

`low`

The starting IIN prefix of the range (inclusive).

Type
str

`high`

The ending IIN prefix of the range (inclusive). If None, only 'low' is used.

Type
Optional[str]

`min_len`

Minimum card number length. Ignored if 'lengths' is provided.

Type
int

`max_len`

Maximum card number length. Ignored if 'lengths' is provided.

Type
int

`lengths`

Explicit list of valid lengths. If provided, overrides min_len/max_len.

Type
Optional[list[int]]

Used to define card validation logic for ranges like: - '400000' to '499999' for Visa - '510000' to '559999' for older MasterCard prefixes

`DINERS: Final[int] = 16`

`DINERS_VALIDATOR: Final = <src.apache_commons_validator_python.routines.code_validator.CodeValidator object>`

Discover Card regular expressions <ul> <li>6011xx (16)</li> <li>644xxx - 65xxxx (16)</li> </ul>

`DISCOVER: Final[int] = 8`

`DISCOVER_REGEX: Final = ['^(6011\\d{12,13})$', '^(64[4-9]\\d{13})$', '^(65\\d{14})$', '^(62[2-8]\\d{13})$']`

`DISCOVER_VALIDATOR: Final = <src.apache_commons_validator_python.routines.code_validator.CodeValidator object>`

`LUHN_VALIDATOR: Final = <src.apache_commons_validator_python.routines.checkdigit.luhn_checkdigit.LuhnCheckDigit object>`

American Express (Amex) Card Validator <ul> <li>34xxxx (15)</li> <li>37xxxx (15)</li> </ul>

`MASTERCARD: Final[int] = 4`

`MASTERCARD_PRE_OCT2016: Final[int] = 64`

`MASTERCARD_REGEX: Final = ['^(5[1-5]\\d{14})$', '^(2221\\d{12})$', '^(222[2-9]\\d{12})$', '^(22[3-9]\\d{13})$', '^(2[3-6]\\d{14})$', '^(27[01]\\d{13})$', '^(2720\\d{12})$']`

`MASTERCARD_VALIDATOR: Final = <src.apache_commons_validator_python.routines.code_validator.CodeValidator object>`

Mastercard Card Validator (pre Oct 2016) @deprecated for use until Oct 2016 only

`MASTERCARD_VALIDATOR_PRE_OCT2016: Final = <src.apache_commons_validator_python.routines.code_validator.CodeValidator object>`

`MAX_CC_LENGTH: Final[int] = 19`

`MIN_CC_LENGTH: Final[int] = 12`

`NONE: Final[int] = 0`

`VISA: Final[int] = 2`

`VISA_VALIDATOR: Final = <src.apache_commons_validator_python.routines.code_validator.CodeValidator object>`

`VPAY: Final[int] = 32`

`VPAY_VALIDATOR: Final = <src.apache_commons_validator_python.routines.code_validator.CodeValidator object>`

`cloneable = False`

`create_range_validator`(*ranges: list[*CreditCardRange*]*, *check_digit*) → CodeValidator

Creates a custom validator that uses a numeric pattern and checks if the prefix and length match any of the provided ranges.

Parameters

- `ranges` – A list of CreditCardRange objects specifying IIN and length rules.
- `check_digit` – A CheckDigit instance used for final validation.

Returns
A CodeValidator that checks card numbers against all provided ranges.

classmethod generic_credit_card_validator() → *CreditCardValidator*

Creates a validator that only checks for numeric card numbers with Luhn validation, using the default min and max length.

Returns
A CreditCardValidator instance using numeric+Luhn check.

classmethod generic_credit_card_validator_with_exact_length(*length: int*) → *CreditCardValidator*

Creates a validator for a specific length, e.g., 16-digit cards.

Parameters
`length` – The exact card length to validate.

Returns
A CreditCardValidator configured to check that length.

classmethod generic_credit_card_validator_with_range(*min_len: int*, *max_len: int*) → *CreditCardValidator*

Creates a validator that only ensures the card is numeric, within a given length range, and passes the Luhn check.

Parameters

- `min_len` – Minimum allowed length.
- `max_len` – Maximum allowed length.

Returns
A CreditCardValidator instance with Luhn check and length bounds.

static is_on(*options: int*, *flag: int*) → bool

Checks if a bitmask flag is enabled in the options.

Parameters

- `options` – Bitmask of all enabled flags.
- `flag` – The specific flag to check.

Returns
True if the flag is enabled in options, False otherwise.

is_valid(*card: str*) → bool

Returns True if the card is valid according to any of the configured card types.

Parameters
`card` – The credit card number as a string.

Returns:
True if the card is valid, False otherwise.

serializable = True

static valid_length(*value_length: int*, *range:* CreditCardRange) → bool

Checks whether a given length is valid based on either an explicit list or a min/max range for the credit card.

Parameters

- `value_length` – The length of the credit card number.
- `range` – The CreditCardRange to validate against.

Returns
True if the length is valid, False otherwise.

`validate`(*card: str*) → str | None

Validates the card and returns the cleaned card number if valid, otherwise None.

Parameters
`card` – The credit card number to validate.

Returns
The card number if valid, or None if invalid.

apache_commons_validator_python.routines.currency_validator module

Module Name: currency_validator.py

Description: Translates apache.commons.validator.routines.CurrencyValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/CurrencyValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.CurrencyValidator.java):

`class apache_commons_validator_python.routines.currency_validator.`**`CurrencyValidator`**(*strict: bool = True*, *allow_fractions: bool = True*)

Bases: *BigDecimalValidator*

Currency Validation and Conversion routines.

This is one implementation of a currency validator that has the following features:

- It is lenient about the presence of the currency symbol.
- It converts the currency to a float.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted big decimal value.

Fraction/decimal values are automatically rounded to the appropriate length.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

`classmethod get_instance()`

Gets the singleton instance of this validator.

Returns
A singleton instance of the validator.

apache_commons_validator_python.routines.date_validator module

Module Name: date_validator.py

Description: Translates apache.commons.validator.routines.DateValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/DateValidator.java

Author: Alicia Chu

License (Taken from apache.commons.validator.routines.DateValidator.java):

Changes:

`class apache_commons_validator_python.routines.date_validator.DateValidator(*, strict: bool = True, date_style: int = 3)`

Bases: `AbstractCalendarValidator`

Date validation and conversion utilities.

This module provides methods to validate and convert string representations of dates into timezone naive *datetime.datetime* objects using various parsing formats and locales.

Supported conversions: - Default format for the default locale - Specified pattern with the default locale - Default format for a specified locale - Specified pattern with a specified locale

All conversion methods allow optional specification of time zones. If this is not provided, the system default will be used.

Validation methods: - *is_valid()*: Checks if a string is a valid date. - *validate()*: Returns a converted *date* object if valid.

Date comparison methods: - *compare_dates(d1, d2)*: Compares day, month, year of two dates. - *compare_weeks(d1, d2)*: Compares week and year of two dates. - *compare_months(d1, d2)*: Compares month and year of two dates. - *compare_quarters(d1, d2)*: Compares quarter and year of two dates. - *compare_years(d1, d2)*: Compares years of two dates.

Formatting methods mirror parsing options and support: - Specified pattern - Format for a specified locale - Format for the default locale

`serializable`

Indicates if the object is serializable.

Type
bool

`cloneable`

Indicates if the object can be cloned.

Type
bool

`VALIDATOR`

The singleton instance of this class

Type
DateValidator

`cloneable = False`

`compare_dates`(*value: datetime*, *compare: datetime*, *time_zone: tzinfo | None = None*) → int

Compare two datetime values by day, month, and year (ignores time component).

Parameters

- **value** (*datetime*) – The first datetime to compare.
- **compare** (*datetime*) – The second datetime to compare against.
- **time_zone** (*Optional[tzinfo]*) – Optional time zone to align both values before comparison.

Returns
0 if the dates are equal, -1 if the first date is earlier, +1 if the first date is later.

Return type
int

`compare_months`(*value: datetime*, *compare: datetime*, *time_zone: tzinfo | None = None*) → int

Compare two datetime values by month and year (ignores day and time).

Parameters

- **value** (*datetime*) – The first datetime to compare.
- **compare** (*datetime*) – The second datetime to compare against.
- **time_zone** (*Optional[tzinfo]*) – Optional time zone to align both values before comparison.

Returns
0 if the month/year are equal, -1 if the first is earlier, +1 if the first is later.

Return type
int

`compare_quarters`(*value: datetime*, *compare: datetime*, *time_zone: tzinfo | None = None*, *month_of_first_quarter: datetime = 1*) → int

Compare two datetime values by quarter and year.

Parameters

- **`value`** (*`datetime`*) – The first datetime to compare.
- **`compare`** (*`datetime`*) – The second datetime to compare against.
- **`time_zone`** (*`Optional[tzinfo]`*) – Optional time zone to align both values before comparison.
- **`month_of_first_quarter`** (*`datetime`*) – Month the first quarter starts (1 = January, 3 = March, etc.).

Returns
0 if quarters are equal, -1 if the first is earlier, +1 if the first is later.

Return type
int

Changes from Java:
month_of_first_quarter defaults to 1 to replace "public int compareQuarters(final Date value, final Date compare, final TimeZone timeZone) {

return compareQuarters(value, compare, timeZone, 1);

}"

`compare_weeks`(*value: datetime*, *compare: datetime*, *time_zone: tzinfo | None = None*) → int

Compare two datetime values by ISO week number and year.

Parameters

- **`value`** (*`datetime`*) – The first datetime to compare.
- **`compare`** (*`datetime`*) – The second datetime to compare against.
- **`time_zone`** (*`Optional[tzinfo]`*) – Optional time zone to align both values before comparison.

Returns
0 if the ISO week/year are equal, -1 if the first is earlier, +1 if the first is later.

Return type
int

`compare_years`(*value: datetime*, *compare: datetime*, *time_zone: tzinfo | None = None*) → int

Compare two datetime values by year only.

Parameters

- **`value`** (*`datetime`*) – The first datetime to compare.
- **`compare`** (*`datetime`*) – The second datetime to compare against.
- **`time_zone`** (*`Optional[tzinfo]`*) – Optional time zone to align both values before comparison.

Returns
0 if years are equal, -1 if the first year is earlier, +1 if the first year is later.

Return type
int

`classmethod get_instance()` → *DateValidator*

Returns the singleton instance of DateValidator. Ensures only one instance is created and reused globally.

Returns
A singleton instance of the class.

Return type
DateValidator

`serializable = True`

`validate`(*value: str*, *pattern: str | None = None*, *locale: str | None = None*, *time_zone: tzinfo | None = None*) → datetime | None

Validates and converts a date string into a datetime object using the provided pattern, locale, and time zone.

Parameters

- **`value`** (*`str`*) – The input string to validate.
- **`pattern`** (*`Optional[str]`*) – The date pattern (e.g., 'yyyy-MM-dd'). If None, a default pattern is used.
- **`locale`** (*`Optional[str]`*) – The locale to apply (e.g., 'en_US'). If None, the system default is used.
- **`time_zone`** (*`Optional[tzinfo]`*) – The timezone to apply to the resulting datetime. If None, no change is made.

Returns
A valid datetime object if parsing succeeds, or None if invalid.

Return type
Optional[datetime]

apache_commons_validator_python.routines.domain_validator module

Module Name: domain_validator.py

Description: Translates apache.commons.validator.routines.DomainValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/DomainValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.DomainValidator.java):

`class apache_commons_validator_python.routines.domain_validator.`**`DomainValidator`**(**args*, ***kwargs*)

Bases: `object`

Domain name validation routines.

This validator provides methods for validating Internet domain names and top-level domains.

Domain names are evaluated according to the standards of RFC1034 section 3 and RFC1123 section 2.1. No accommodation is provided for the specialized needs of other applications; if the domain name has been URL-encoded, for example, validation will fail even though the equivalent plaintext version of the same name would have passed.

Validation is also provided for top-level domains (TLDs) as defined and maintained by the Internet Assigned Numbers Authority (IANA):

- is_valid_infrastructure_tld(): validates infrastructure TLDs (.arpa, etc.).
- is_valid_generic_tld(): validates generic TLDs (.com, .org, etc.).
- is_valid_country_code_tld(): validates country code TLDs (.us, .uk, .cn, etc.).

(NOTE: This class does not provide IP address lookup for domain names or methods to ensure that a given domain name matches a specific IP; see inet_address_validator.py for that functionality.)

`class ArrayType`(*value*, *names=<not given>*, *values, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `Enum`

Enum used by update_tld_override(ArrayType, list[str]) to determine which override list to update/fetch.

`- GENERIC_PLUS`
Update (or get a copy of) the GENERIC_TLDS_PLUS table containing additional generic TLDs.

`- GENERIC_MINUS`
Update (or get a copy of) the GENERIC_TLDS_MINUS table containing deleted generic TLDs.

`- COUNTRY_CODE_PLUS`
Update (or get a copy of) the COUNTRY_code_tlds_PLUS table containing additional country code TLDs.

`- COUNTRY_CODE_MINUS`
Update (or get a copy of) the COUNTRY_code_tlds_MINUS table containing deleted country code TLDs.

`- GENERIC_RO`
Gets a copy of the generic TLDS table.

`- COUNTRY_CODE_RO`
Gets a copy of the country code table.

`- INFRASTRUCTURE_RO`
Gets a copy of the infrastructure table.

`- LOCAL_RO`
Gets a copy of the local table.

`- LOCAL_PLUS`
Update (or get a copy of) the LOCAL_TLDS_PLUS table containing additional local TLDs.

`- LOCAL_MINUS`
Update (or get a copy of) the LOCAL_TLDS_MINUS table containing deleted local TLDs.

`COUNTRY_CODE_MINUS = 4`

`COUNTRY_CODE_PLUS = 3`

`COUNTRY_CODE_RO = 6`

`GENERIC_MINUS = 2`

`GENERIC_PLUS = 1`

`GENERIC_RO = 5`

`INFRASTRUCTURE_RO = 7`

`LOCAL_MINUS = 10`

`LOCAL_PLUS = 9`

`LOCAL_RO = 8`

class Item(*type*, *values: list[str]*)

Bases: `object`

Used to specify overrides when creating a new class.

property allow_local

Whether or not this instance allows local addresses.

Returns
True if local addresses are allowed.

`cloneable = False`

classmethod get_instance(*allow_local: bool = False*, *items=None*)

Returns the singleton instance of this validator, with local validation if required (not by default). The user can provide a list of Item entries which can be used to override the generic and country code lists. Note that any such entries override values provided by the update_tld_override(ArrayType, str) method. If an entry for a particular type is not provided, then the class override (if any) is retained.

Parameters

- **allow_local** (*bool*) – If local addresses be considered valid.
- **items** (*list[Item]*) – List of Item entries.

Returns
The singleton instance of this validator.

get_overrides(*table: ArrayType*)

Gets a copy of an instance level internal list.

Parameters
table (ArrayType) – The ArrayType (any of the enum values).

Returns
A copy of the list. Throws a *ValueError* if the table type is unexpected, for example, GENERIC_RO.

static get_tld_entries(*table: ArrayType*)

Gets a copy of a class level internal list.

Parameters
table (ArrayType) – The ArrayType (any of the enum values).

Returns

A copy of the list. Throws a *ValueError* if the table type is unexpected
(should not happen).

`is_valid`(*domain: str*)

Returns *True* if the specified domain parses as a valid domain name with a recognized top-level domain. The parsing is case-insensitive.

Parameters
`domain` (*str*) – The parameter to check for domain name syntax.

Returns
True if the parameter is a valid domain name.

`is_valid_country_code_tld`(*cc_tld: str*)

Returns True if the specified string matches any IANA-defined country code top-level domain. Leading dots are ignored if present. The search is case- insensitive.

Parameters
`cc_tld` (*str*) – The parameter to check for country code TLD status, must not be *None*.

Returns
True if the parameter is a country code TLD.

`is_valid_generic_tld`(*g_tld: str*)

Returns *True* if the specified string matches any IANA-defined generic top- level domain. Leading dots are ignored if present. The search is case- insensitive.

Parameters
`g_tld` (*str*) – The parameter to check for generic TLD status, must not be *None*.

Returns
True if the parameter is a generic TLD.

`is_valid_infrastructure_tld`(*i_tld: str*)

Returns *True* if the specified string matches any IANA-defined infrastructure top-level domain. Leading dots are ignored if present. The search is case- insensitive.

Parameters
`i_tld` (*str*) – The parameter to check for infrastructure TLD status, must not be *None*.

Returns
True if the parameter is an infrastructure TLD.

`is_valid_local_tld`(*l_tld: str*)

Returns *True* if the specified string matches any widely used "local" domains (localhost or localdomain). Leading dots are ignored if present. The search is case-insensitive.

Parameters
`l_tld` (*str*) – The parameter to check for local TLD status, must not be *None*.

Returns
True if the parameter is a local TLD.

`is_valid_tld`(*tld: str*)

Returns *True* if the specified string matches any IANA-defined top-level domain. Leading dots are ignored if present. The search is case-insensitive.

If allow_local is *True*, the TLD is checked using is_valid_local_tld(str). The TLD is then checked against is_valid_infrastructure_tld(str), is_valid_generic_tld(str) and is_valid_country_code_tld(str).

Parameters
`tld` (*str*) – The parameter to check for TLD status, must not be *None*.

Returns
True if the parameter is a TLD.

serializable = True

static unicode_to_ascii(*input: str*)

Converts potentially Unicode input to punycode. If conversion fails, returns the original input.

Parameters

`input` (*str*) – The string to convert, must not be *None*.

Returns

The converted input, or original input if conversion fails.

static update_tld_override(*table: ArrayType*, *tlds: list[str]*)

Update one of the TLD override arrays. This must only be done at program startup, before any instances are accessed using get_instance.

For example: updateTLDOverride(ArrayType.GENERIC_PLUS, "apache") To clear an override list, provide an empty list.

Parameters

- **table** (*ArrayType*) – The table to update, see ArrayType. Must be one of the following: - COUNTRY_CODE_MINUS - COUNTRY_CODE_PLUS - GENERIC_MINUS - GENERIC_PLUS - LOCAL_MINUS - LOCAL_PLUS
- **tlds** (*list[str]*) – The list of TLDs, must not be *None*.

Returns

Throws an *Exception* if the validator is in use. Throws a *ValueError* if one of the read-only tables is requested.

apache_commons_validator_python.routines.double_validator module

Module Name: double_validator.py

Description: Translates apache.commons.validator.routines.DoubleValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/DoubleValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.DoubleValidator.java):

class apache_commons_validator_python.routines.double_validator.**DoubleValidator**(*strict: bool = True*, *format_type: int = 0*)

Bases: *AbstractNumberValidator*

Double Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to a double to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted double value.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

`classmethod get_instance()`

Gets the singleton instance of this validator.

Returns
A singleton instance of the validator.

`validate`(*value: str*, *pattern: str = None*, *locale=None*)

Validate/convert a double using the optional pattern and/or locale.

Parameters

- **value** (*str*) – The value validation is being performed on.
- **pattern** (*str*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- **locale** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns
The parsed double (as a float) if valid or *None* if invalid.

apache_commons_validator_python.routines.email_validator module

Module Name: email_validator.py

Description: Translates apache.commons.validator.routines.EmailValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/EmailValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.EmailValidator.java):

class `apache_commons_validator_python.routines.email_validator.`**`EmailValidator`**(*allow_local: bool = False*, *allow_tld: bool = False*, *domain_validator: DomainValidator = None*)

Bases: `object`

Perform email validations.

Based on a script by Sandeep V. Tamhankar (https://javascript.internet.com).

This implementation is not guaranteed to catch all possible errors in an email address.

`cloneable = False`

`classmethod get_instance`(*allow_local: bool = False*, *allow_tld: bool = False*)

Returns the Singleton instance of this validator, with local and/or TLD validation as required.

Parameters

- **`allow_local`** (*`bool`*) – Should local addresses be considered valid? Default is *False*.
- **`allow_tld`** (*`bool`*) – Should TLDs be allowed? Default is *False*.

Returns

Singleton instance of this validator.

`is_valid`(*email: str*)

Checks if a field has a valid e-mail address.

Parameters

`email` (*`str`*) – The value validation is being performed on. A value of *None* is considered invalid.

Returns

True if the email address is valid.

`serializable = True`

apache_commons_validator_python.routines.float_validator module

Module Name: float_validator.py

Description: Translates apache.commons.validator.routines.FloatValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/FloatValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.FloatValidator.java):

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to You under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http:#www.apache.org/licenses/LICENSE-2.0

`class apache_commons_validator_python.routines.float_validator.`**`FloatValidator`**`(`*strict: bool = True*, *format_type: int = 0*`)`

Bases: `AbstractNumberValidator`

Float Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to a float to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted float value.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

`FLOAT_MIN`

The minimum value of a float.

Type

Final[float]

`FLOAT_MAX`

The maximum value of a float.

Type

Final[float]

`FLOAT_MAX: Final[float] = 3.402823466e+38`

`FLOAT_MIN: Final[float] = 1.175494351e-38`

`classmethod get_instance()`

Gets the singleton instance of this validator.

Returns

A singleton instance of the validator.

`validate``(`*value: str*, *pattern: str = None*, *locale=None*`)`

Validate/convert a float using the optional pattern and/or locale.

Parameters

- **value** (*str*) – The value validation is being performed on.

- **`pattern`** (*`str`*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- **`locale`** (*`str`*) – The (optional) locale to use for the format, defaults to the system default.

Returns
The parsed float if valid or *None* if invalid.

apache_commons_validator_python.routines.inet_address_validator module

Module Name: inet_address_validator.py

Description: Translates apache.commons.validator.routines.InetAddressValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/InetAddressValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.InetAddressValidator.java):

`class apache_commons_validator_python.routines.inet_address_validator.`**`InetAddressValidator`**

Bases: `object`

Inet Address validation and conversion routines.

This class provides methods to validate a candidate IP address.

This class is a Singleton; you can retrieve the instance via the get_instance() method.

`cloneable = False`

`classmethod` **`get_instance`**()

Returns the singleton instance of this validator.

Returns
The singleton instance of this validator.

`is_valid`(*inet_address: str*)

Checks if the specified string is a valid IPv4 or IPv6 address.

Parameters
`inet_address` (*`str`*) – The string to validate.

Returns
True if the string validates as an IP address.

`is_valid_inet4_address`(*inet4_address: str*)

Validates an IPv4 address. Returns *True* if valid.

Parameters
`inet4_address` (*str*) – The IPv4 address to validate.

Returns
True if the argument contains a valid IPv4 address.

`is_valid_inet6_address`(*inet6_address: str*)

Validates an IPv6 address. Returns true if valid.

Parameters
`inet6_address` (*str*) – The IPv6 address to validate.

Returns
True if the argument contains a valid IPv6 address.

`serializable = True`

apache_commons_validator_python.routines.integer_validator module

Module Name: integer_validator.py

Description: Translates apache.commons.validator.routines.IntegerValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/IntegerValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.IntegerValidator.java):

`class apache_commons_validator_python.routines.integer_validator.IntegerValidator`(*strict: bool = True*, *format_type: int = 0*)

Bases: `AbstractNumberValidator`

Integer Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to an integer to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted int value.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

`INT_MIN`

The minimum value of an int (-2^31).

Type
int

`INT_MAX`

The maximum value of an int (2^31 - 1).

Type
int

`INT_MAX: Final[int] = 2147483647`

`INT_MIN: Final[int] = -2147483648`

`classmethod` **`get_instance()`**

Gets the singleton instance of this validator.

Returns
A singleton instance of the validator.

`validate`(*value: str*, *pattern: str = None*, *locale=None*)

Validate/convert an integer using the optional pattern and/or locale.

Parameters

- **value** (*str*) – The value validation is being performed on.
- **pattern** (*str*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- **locale** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns
The parsed int if valid or *None* if invalid.

apache_commons_validator_python.routines.isbn_validator module

Module Name: isbn_validator.py

Description:
Translates apache.commons.validator.routines.ISBNValidator.java This module provides a class *ISBNValidator* for validating ISBN-10 and ISBN-13 codes. It also supports converting ISBN-10 codes to ISBN-13 if the *convert* attribute is set to *fov*. The class uses regular expressions to validate the formats and check digits for ISBN-10 and ISBN-13 codes. It also allows ISBN-10 codes to be converted to ISBN-13 using specific conversion logic.

Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/ISBNValidator.java

Author: Juji Lau

License:

Changes:

- Removed *getInstance()* method, which supports singleton behavior in the Java version. Singleton behavior is implicit in this Python version.
- Added a setter for the *convert* property. In the original Java version, *getInstance(convert)* provided a way to configure this, but it has been replaced with a direct attribute.
- removed ISBN_VALIDATOR and ISBN_VALIDATOR_NO_CONVERT from the java version (idk how to implement)

`class apache_commons_validator_python.routines.isbn_validator.`**`ISBNValidator`**`(`*convert: bool = True*`)`

Bases: `object`

A validator class for ISBN-10 and ISBN-13 codes.

This class validates whether a code is a valid ISBN-10 or ISBN-13 and provides the ability to convert ISBN-10 codes to ISBN-13 if the *convert* property is set to `True`.

`convert`

If True, all ISBN-10 codes will be converted to ISBN-13.

Type
bool

`isbn10_validator`

Validator instance for ISBN-10 codes.

Type
CodeValidator

`isbn13_validator`

Validator instance for ISBN-13 codes.

Type
CodeValidator

`serializable`

Indicates if the object is serializable.

Type
bool

`cloneable`

Indicates if the object can be cloned.

Type
bool

Constants:

ISBN_10_LEN (int): The length of an ISBN-10 code. SEP (str): The separator used in ISBN code formats (either hyphens or spaces). GROUP (str): Regular expression pattern for the group portion of the ISBN code. PUBLISHER (str): Regular expression pattern for the publisher portion of the ISBN code. TITLE (str): Regular expression pattern for the title portion of the ISBN code. ISBN10_REGEX (str): Regular expression pattern for ISBN-10 validation. ISBN13_REGEX (str): Regular expression pattern for ISBN-13 validation.

`ISBN10_REGEX = '^(?:(\\d{9}[0-9X])|(?:(\\d{1,5})(?:\\-|\\s)(\\d{1,7})(?:\\-|\\s)(\\d{1,6})(?:\\-|\\s)([0-9X])))$'`

`ISBN13_REGEX = '^(978|979)(?:(\\d{10})|(?:(?:\\-|\\s)(\\d{1,5})(?:\\-|\\s)(\\d{1,7})(?:\\-|\\s)(\\d{1,6})(?:\\-|\\s)([0-9])))$'`

`cloneable = False`

`property convert`

Returns the convert attribute.

`convert_to_isbn13`(*isbn10: str*) → str

Convert an ISBN-10 code to an ISBN-13 code.

Parameters
`isbn10` (*str*) – The ISBN-10 code to convert. Must be a valid ISBN-10 with NO formatting characters.

Returns
A converted ISBN-13 code, or None if the ISBN-10 code is not valid.

Raises
`ValueError if isbn10 is invalid or has formatting errors.` –

`classmethod get_instance`(*convert: bool = True*)

Gets the singleton instance of the ISBN validator specifying whether ISBN-10 codes should be converted to ISBN-13.

Parameters
`convert` (*bool*) – `True` if valid ISBN-10 codes should be converted to ISBN-13 codes. `False` if valid ISBN-10 codes should be returned unchanged.

Returns
A singleton instance of the ISBN validator.

`is_valid`(*code: str*) → bool

Checks if code is either a valid ISBN-10 or ISBN-13 code.

Parameters
`code` (*str*) – The code to check.

Returns
True if the code is a valid ISBN-10 or ISBN-13 code, otherwise false.

`is_valid_isbn10`(*code: str*) → bool

Checks if code is a valid ISBN-10 code.

Parameters
`code` (*str*) – The ISBN-10 code to check.

Returns
True if the code is a valid ISBN-10 code, False otherwise.

is_valid_isbn13(*code: str*) → bool

Checks if code is a valid ISBN-13 code.

Parameters
`code` (*str*) – The ISBN-13 code to check.

Returns
True if the code is a valid ISBN-13 code, otherwise false.

property isbn10_validator

Returns the isbn10_validator attribute.

property isbn13_validator

Returns the isbn13_validator attribute.

serializable = True

validate(*code: str*) → str

Checks if code is either a valid ISBN-10 or ISBN-13 code. If valid, this method returns the ISBN code with formatting characters removed (i.e. space or hyphen). Converts an ISBN-10 codes to ISBN-13 if convertToISBN13 is true.

Parameters
`code` (*str*) – The code to validate.

Returns
A valid ISBN code if valid, otherwise None.

validate_isbn10(*code: str*) → str

Checks if code is a valid ISBN-10 code. If valid, this method returns the ISBN-10 code with formatting characters removed (i.e. space or hyphen).

Parameters
`code` (*str*) – The code to validate.

Returns
A valid ISBN-10 code if valid, otherwise None.

validate_isbn13(*code: str*) → str

Checks if code is a valid ISBN-13 code. If valid, this method returns the ISBN-13 code with formatting characters removed (i.e. space or hyphen).

Parameters
`code` (*str*) – The code to validate.

Returns
A valid ISBN-13 code if valid, otherwise None.

apache_commons_validator_python.routines.isin_validator module

Module Name: isin_validator.py Description:

Translates apache.commons.validator.routines.ISINValidator.java

Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/ISINValidator.java

Author: Alicia Chu

Changes: - Locale.getISOLanguages() -> pycountry.countries.alpha_2 (ISO 3166-1 alpha-2 – two-letter country codes which are used most prominently for the Internet's country code top-level domains)

`class apache_commons_validator_python.routines.isin_validator.`**`ISINValidator`**(*check_country_code: bool*)

Bases: `object`

Validates ISIN (International Securities Identifying Number) codes.

`get_instance`(*check_country_code*)

Returns a singleton instance of the validator.

`is_valid`(*code*)

Checks if the given code is a valid ISIN.

`validate`(*code*)

Returns the code if valid, otherwise None.

`cloneable: Final[bool] = False`

`classmethod get_instance`(*check_country_code: bool*) → *ISINValidator*

Gets the singleton instance of ISINValidator.

Parameters

`check_country_code` (*`bool`*) – Whether to enforce country code validation.

Returns

The appropriate singleton validator instance.

Return type

ISINValidator

`is_valid`(*code: str* | *None*) → bool

Checks if the provided ISIN code is valid.

Parameters

`code` (*`str`*) – The code to validate.

Returns

True if valid, False otherwise.

Return type

bool

`serializable: Final[bool] = True`

`validate`(*code: str* | *None*) → str | None

Validates and returns the ISIN code if valid.

Parameters
`code` (*str*) – The ISIN code to validate.

Returns
The valid code or None if invalid.

Return type
Optional[str]

apache_commons_validator_python.routines.long_validator module

Module Name: long_validator.py

Description: Translates apache.commons.validator.routines.LongValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/LongValidator.java

Author: Jessica Breuhaus

`class apache_commons_validator_python.routines.long_validator.`**`LongValidator`**(*strict: bool = True*, *format_type: int = 0*)

Bases: `AbstractNumberValidator`

Long Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to a long to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted long value.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

`LONG_MIN`

The minimum value of a long (-2^63).

Type
int

`LONG_MAX`

The maximum value of a long (2^63 - 1).

Type
int

`LONG_MAX: Final[int] = 9223372036854775807`

`LONG_MIN: Final[int] = -9223372036854775808`

`classmethod get_instance()`

Gets the singleton instance of this validator.

Returns
A singleton instance of the validator.

`validate`(*value: str*, *pattern: str = None*, *locale=None*)

Validate/convert a long using the optional pattern and/or locale.

Parameters

- `value` (*str*) – The value validation is being performed on.
- `pattern` (*str*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- `locale` (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns
The parsed long (as an int) if valid or *None* if invalid.

apache_commons_validator_python.routines.percent_validator module

Module Name: percent_validator.py

Description: Translates apache.commons.validator.routines.PercentValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/PercentValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.PercentValidator.java):

class apache_commons_validator_python.routines.percent_validator.**PercentValidator**(*strict: bool = True*)

Bases: *BigDecimalValidator*

Percent Validation and Conversion routines.

This is one implementation of a percent validator that has the following features:

- It is lenient about the presence of the currency symbol.
- It converts the percent to a float.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted big decimal value.

Fraction/decimal values are automatically rounded to the appropriate length.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

classmethod get_instance()

Gets the singleton instance of this validator.

Returns
A singleton instance of the validator.

apache_commons_validator_python.routines.regex_validator module

Module Name: regex_validator.py Description: Translates apache.commons.validator.routines.RegexValidator.java

Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/RegexValidator.java Paraphrased from apache.commons.validator.routines.RegexValidator:

Regular Expression validation (using Python's built-in `re` module).

Constructs the validator either for a single regular expression or a set (list) of regular expressions. By default, validation is *case sensitive* but constructors are provided to allow *case-insensitive* validation.

Example:

To create a validator which does case in-sensitive validation for a set of regular expressions:
` regexs = "some_str" validator = RegexValidator(regexs, case_sensitive=False) `

Validate returning a boolean (`True` or `False`):
` valid = validator.is_valid(some_value) `

Validate returning an aggregated String of the matched groups:
` result = validator.validate(some_value) `

Validate and return the matched groups as a list of strings:
` result = validator.match(some_value) `

Note:
Patterns are matched against the entire input.

Thread Safety:
Compiled regex patterns are cached and can be used in multi-threaded environments safely.

Author: Juji Lau License (Taken from apache.commons.validator.routines.RegexValidator):

Changes:

- Java uses the package `Pattern` for regular expressions. This Python file uses the `re` package.
- **Removed:**
  - `CASE_SENSITIVE`
    It was private constant (so no outside code calling). Instead, we passed `case_sensitive` as an argument to __init__(), setting it to True by default.
  - `toCompileFlags()`
    We only have one flag to keep track of (`re.IGNORECASE`).
- **Substitutions (Java -> Python):**
  `java.util.regex` -> `re` Regex package `regex.compile()` -> `re.compile()` Compiles a Pattern `regex.Pattern` -> `re.Pattern` `Pattern.CASE_INSENSITIVE` -> `re.INGORECASE` Flag to ignore case when pattern matching. `Pattern.pattern` -> `Pattern.pattern` Field that represents the pattern regex as a string. `Pattern.matcher(value).matches()` -> `Pattern.fullmatch(value)` Matches the entire value against the pattern.

  Java:
  `Pattern.matcher(value)` Creates a `Matcher` object that matches the entire value against the pattern. `matches()` Returns `True` iff the entire value matches the regex pattern.

  Python:
  `Pattern.fullmatch()` Creates a `Match` object that matches the entire value against the pattern. None if there is no match.

  `regex.Matcher` -> `re.Match` Object created by calling method(s) on `Pattern`. `Matcher.groups()` -> `Match.groups()` List of all the matches in the string to the pattern regex. `java.lang.Object.clone()` -> `copy.copy()` For shallow copies

`class apache_commons_validator_python.routines.regex_validator.`**`RegexValidator`**`(`*regexs: str | list[str], case_sensitive: bool = True*`)`

Bases: `object`

A regular expression validator using Python's *re* module.

Supports validation against one or multiple regex patterns, with an option for case insensitivity.

`patterns`

Compiled regex patterns.

Type
list[Pattern]

`serializable`

Indicates if the object is serializable.

Type
bool

`cloneable`

Indicates if the object can be cloned.

Type
bool

`cloneable = False`

`is_valid`(*value: str*) → bool

Validates a value against the set of regular expressions.

Parameters
`value` (*`str`*) – The value to validate.

Returns
`True` if any pattern fully matches `value`, else `False`.

`match`(*value: str*) → list[str] | None

Matches the input value against the regex patterns and returns matched groups.

Parameters
`value` (*`str`*) – The input string to validate.

Returns
A list of matched groups if valid; otherwise `None`.

Return type
list[str] | None

`property patterns: list[Pattern]`

Returns a shallow copy of the class attribute patterns.

Note: Since we return a shallow copy of self.__patterns, when we referecne self.__patterns in this class, we use self.__patterns to avoid making a new shallow copy at each reference.

`serializable = True`

`validate`(*value: str*) → str | None

Matches the input value and returns the concatenated matched groups.

Parameters
`value` (*`str`*) – The input string to validate.

Returns
Concatenated matched groups if valid; otherwise `None`.

Return type
str | None

apache_commons_validator_python.routines.short_validator module

Module Name: short_validator.py

Description: Translates apache.commons.validator.routines.ShortValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/ShortValidator.java

Author: Jessica Breuhaus

`class apache_commons_validator_python.routines.short_validator.`**`ShortValidator`**`(`*strict: bool = True*, *format_type: int = 0*`)`

Bases: `AbstractNumberValidator`

Short Validation and Conversion routines.

This validator provides a number of methods for validating/converting a string value to a short to parse either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

Use the is_valid() method to just validate or one of the validate() methods to validate and receive a converted short value.

So that the same mechanism used for parsing an input value for validation can be used to format output, corresponding format() methods are also provided. That is you can format either:

- using the default format for the default locale.
- using a specified pattern with the default locale.
- using the default format for a specified locale.
- using a specified pattern with a specified locale.

`SHORT_MAX: Final[int] = 32767`

`SHORT_MIN: Final[int] = -32768`

`classmethod get_instance()`

Gets the singleton instance of this validator.

Returns

A singleton instance of the validator.

`validate`(*value: str*, *pattern: str = None*, *locale=None*)

Validate/convert a short using the optional pattern and/or locale.

Parameters

- **`value`** (*str*) – The value validation is being performed on.
- **`pattern`** (*str*) – The (optional) regex pattern used to validate the value against, or the default for the locale if *None*.
- **`locale`** (*str*) – The (optional) locale to use for the format, defaults to the system default.

Returns

The parsed short (as an int) if valid or *None* if invalid.

apache_commons_validator_python.routines.time_validator module

Module Name: time_validator.py

Description: Translates apache.commons.validator.routines.TimeValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/TimeValidator.java

Author: Juji Lau

License (Taken from apache.commons.validator.routines.TimeValidator.java):

Changes:

class `apache_commons_validator_python.routines.time_validator.`**`TimeValidator`**(**, *strict: bool = True*, *time_style: int = 3*)

Bases: `AbstractCalendarValidator`

Time validation and conversion utilities.

This module provides methods to validate and convert string representations of times into *datetime.time* objects using various parsing formats and locales.

Supported conversions: - Default format for the default locale - Specified pattern with the default locale - Default format for a specified locale - Specified pattern with a specified locale

All conversion methods (*validate()*) allow optional specification of time zones. If this is not provided, the system default will be used.

Alternatively, the *CalendarValidator.adjust_to_timezone()* method can be used to adjuat the *tzinfo* of the *datetime* object afterwards.

Once a value has been successfully converted the following methods can be used to perform various time comparison checks:

Date comparison methods: - *compare_time(d1, d2)*: Compares the hours, minutes, seconds, and milliseconds of two datettimes. - *compare_seconds(d1, d2)*: Compares the hours, minutes, and seconds of two times. -

compare_minutes(d1, d2): Compares hours and minutes of two times. - *compare_hours(d1, d2)*: Compares the hours of two times

Formatting methods mirror parsing options and support: - Specified pattern - Format for a specified locale - Format for the default locale

`serializable`

Indicates if the object is serializable.

Type
bool

`cloneable`

Indicates if the object can be cloned.

Type
bool

`VALIDATOR`

The singleton instance of this class

Type
TimeValidator

`cloneable = False`

`compare_hours`(*value: datetime*, *compare: datetime*) → int

Compare two datetime values by hours (ignores date component).

Parameters

- **value** (*datetime*) – The first datetime to compare.
- **compare** (*datetime*) – The second datetime to compare against.

Returns
0 if the hours are equal, -1 if the hour of value is less than the hour of compare, +1 if the hour of value is greater than the hour of compare.

`compare_minutes`(*value: datetime*, *compare: datetime*) → int

Compare two datetime values by hours and minutes (ignores the date component).

Parameters

- **value** (*datetime*) – The first datetime to compare.
- **compare** (*datetime*) – The second datetime to compare against.

Returns
0 if the hour/minutes are equal, -1 if the minute of value is less than the minute of compare, +1 if the minute of value is greater than the minute of compare.

`compare_seconds`(*value: datetime*, *compare: datetime*) → int

Compare two datetime values by hours, minutes, and seconds.

Parameters

- **value** (*datetime*) – The first datetime to compare.
- **compare** (*datetime*) – The second datetime to compare against.

Returns
0 if the hour/minutes/seconds are equal, -1 if the seconds of value is less than the seconds of compare, +1 if the seconds of value is greater than the seconds of compare.

`compare_time`(*value: datetime*, *compare: datetime*) → int

Compare two datetime values by absolute time (hour, minute, second, and microscecond).

Parameters

- **value** (*datetime*) – The first datetime to compare.
- **compare** (*datetime*) – The second datetime to compare against.

Returns

0 if the times are equal -1 if the time represented by value is less than (earlier) the time of compare. +1 if the time represented by value is greater than (later) the time of compare.

Changes from Java:

Java compares the *millisecond* of two `Calendar` object. Python's `datetime` does not have milliseconds. The finest unit of time in a `datetime.time` is the *microsecond*. This implementation compares milliseconds to determine the earlier time.

`classmethod get_instance`() → *TimeValidator*

Returns the singleton instance of TimeValidator. Ensures only one instance is created and reused globally.

Returns

A singleton instance of the class.

Return type

TimeValidator

`serializable = True`

`validate`(***, *value: str*, *pattern: str | None = None*, *locale: str | None = None*, *time_zone: tzinfo | None = None*) → datetime | None

Validates and converts a time string into a datetime object using the provided pattern, locale, and time zone. The datetime will represent the time string elapsed since the epoch (year, month, day will be set to 1970, 1, 1 respectively.)

Parameters

- **value** (*str*) – The input string to validate.
- **pattern** (*Optional[str]*) – The date pattern (e.g., 'yyyy-MM-dd'). If None, a default pattern is used.
- **locale** (*Optional[str]*) – The locale to apply (e.g., 'en_US'). If None, the system default is used.
- **time_zone** (*Optional[tzinfo]*) – The timezone to apply to the resulting datetime. If None, no change is made.

Returns

A datetime object representing the time-string elapsed since the the Epoch, if parsing succeeds. `None` if invalid.

Return type

Optional[datetime]

apache_commons_validator_python.routines.url_validator module

Module Name: url_validator.py

Description: Translates apache.commons.validator.routines.UrlValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/UrlValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.UrlValidator.java):

`class apache_commons_validator_python.routines.url_validator.UrlValidator`(*schemes: list[str] = None*, *authority_validator:* RegexValidator *= None*, *options: int = 0*, *domain_validator:* DomainValidator *= None*)

Bases: `object`

URL Validation routines.

Behavior of validation is modified by passing in options:

- ALLOW_2_SLASHES: [FALSE] Allows double '/' characters in the path component.
- **NO_FRAGMENT: [FALSE] By default fragments are allowed, if this option is included**
  then fragments are flagged as illegal.
- **ALLOW_ALL_SCHEMES: [FALSE] By default only http, https, and ftp are considered valid schemes.**
  Enabling this option will let any scheme pass validation.

Originally based in on php script by Debbie Dyer, validation.php v1.2b, Date: 03/07/02, https://javascript.internet.com. However, this validation now bears little resemblance to the php original.

Example of usage: Construct a UrlValidator with valid schemes of "http", and "https".

schemes = {"http","https"}. url_validator = UrlValidator(schemes); if (url_validator.is_valid("ftp://foo.bar.com/")):

print("URL is valid")

else:
print("URL is invalid")

Will return *False*.

If instead the default constructor is used.
url_validator = UrlValidator(); if (url_validator.is_valid("ftp://foo.bar.com/")):

print("URL is valid")

else:
print("URL is invalid")

Will return *True*.

See "http://www.ietf.org/rfc/rfc2396.txt" Uniform Resource Identifiers (URI): Generic Syntax

`- ALLOW_ALL_SCHEMES`
Allow all validly formatted schemes to pass validation instead of supplying a set of valid schemes.

Type
int

`- ALLOW_2_SLASHES`
Allow two slashes in the path component of the URL.

Type
int

`- NO_FRAGMENTS`
Disallow any URL fragments.

Type
int

`- ALLOW_LOCAL_URLS`
Allow local URLs, enabling a broad-brush check.

Type
int

`ALLOW_2_SLASHES: Final[int] = 2`
Allow two slashes in the path component of the URL.

`ALLOW_ALL_SCHEMES: Final[int] = 1`
Allows all validly formatted schemes to pass validation instead of supplying a set of valid schemes.

`ALLOW_LOCAL_URLS: Final[int] = 8`
//machine/ . This enables a broad-brush check, for complex local machine name validation requirements you should create your validator with RegexValidator instead UrlValidator(RegexValidator, int)

Type
Allow local URLs, such as https

Type
//localhost/ or https

`NO_FRAGMENTS: Final[int] = 4`
Enabling this options disallows any URL fragments.

`cloneable = False`

`classmethod get_instance()`
Returns the singleton instance of this class with default schemes and options.

Returns
Singleton instance with default schemes and options.

is_valid(*value: str*)

Checks if a field has a valid URL address.

Note that the method calls is_valid_authority() which checks that the domain is valid.

Parameters
`value` – The value validation is being performed on. *None* is considered an invalid value.

Returns
True if the URL is valid.

serializable = True

Module contents

apache_commons_validator_python.util package

Submodules

apache_commons_validator_python.util.datetime_helpers module

TODO: insert module description

class apache_commons_validator_python.util.datetime_helpers.**JavaToPyLocale**

Bases: `object`

Wrapper class to convert Java's locales to a Python locale string.

GERMAN: str = 'de'

GERMANY: str = 'de_DE'

UK: str = 'en_GB'

US: str = 'en_US'

apache_commons_validator_python.util.datetime_helpers.**date_get_time**(*dt: datetime*) → float

Python wrapper for Java's `Date.getTime()` function. Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this `datetime` object.

Note Java's validator calculates the time since the epoch using a `Date` object. However, Java's `Date` is in the processes of deprecating all references to time fields (e.g. hour, minute, second, etc.). In this project, even though we substitute python's `date` for Java's `Date`, we will use Python's `date` does not store time fields, hence we will be using a `datetime` for this.

Parameters
`dt` (*datetime*) – The Python date to get thet time from.

Returns
00:00 GMT represented by dt.

Return type
The number of milliseconds since January 1, 1970, 00

apache_commons_validator_python.util.datetime_helpers.**fuzzy_parse**(**, value: str, pattern: str, locale: str, settings: dict*) → datetime

Uses `dateparser.parse()` to parse a datetime given a value string, locale, and pattern. This is a last resort, if all else fails, because dateparser.parse() is too loose; it allows differing value strings to be parsed. We still use it because it respects locales the best.

Parameters
`TODO`

Returns
TODO

`apache_commons_validator_python.util.datetime_helpers.`**`get_default_locale`**`()` → str

Gets the system's default locale (*en_US*).

`apache_commons_validator_python.util.datetime_helpers.`**`get_default_tzinfo`**`()` → tzinfo

Gets the system's default timezone.

`apache_commons_validator_python.util.datetime_helpers.`**`get_tzname`**`(`*timezone: tzinfo*`)` → str

Returns the name of the timezone (same as `datetime.tzname`). *tzinfo* does not have a name field, so this function is nessescary to get the name of a lone tzinfo objct.

Parameters
`timezone` (*`tzinfo`*) – The tzinfo object that we want the name of

Returns
The name of timezone as a string.

`apache_commons_validator_python.util.datetime_helpers.`**`ldml2strpdate`**`(`*value: str*, *style_format: str = 'short'*, *locale: str = None*`)` → datetime

Parses the value to a `datetime` based on the style_format and locale. Uses system default if the locale is `None`.

Parameters

- **`value`** (*`str`*) – The string to parse into a datetie
- **`style_format`** (*`str`*) – The style of the `value` string passed in ('short' by default.) One of: 'short', 'medium', 'long', or 'full'
- **`locale`** (*`str`*) – The locale of the value string.

Returns
The parsed `datetime` from the value string. `None` if the value string is unparseable with the given locale.

`apache_commons_validator_python.util.datetime_helpers.`**`ldml2strptime`**`(`*value: str*, *style_format: str = 'short'*, *locale: str = None*`)` → datetime

Parses the value to a `datetime` based on the style_format and locale. Uses system default if the locale is `None`.

Parameters

- **`value`** (*`str`*) – The string to parse into a `datetime`
- **`style_format`** (*`str`*) – The style of the `value` string passed in ('short' by default.) One of: 'short', 'medium', 'long', or 'full'
- **`locale`** (*`str`*) – The locale of the value string.

Returns
The parsed `datetime` from the value string. `None` if the value string is unparseable with the given locale.

`apache_commons_validator_python.util.datetime_helpers.`**`ldml_to_strptime_format`**(*java_input: str*) → str

Convert a Java SimpleDateFormat pattern into an equivalent Python strftime pattern.

Steps:

1. Scan the input string for substrings like 'yyyy', 'MM', 'dd', etc.
2. For each match, look up its replacement in JAVA_TO_PY.
3. Produce a new format string with all tokens swapped out.

Example

java_fmt = "yyyy-MM-dd'T'HH:mm:ss.SSSZ" returns "%Y-%m-%dT%H:%M:%S.%f%z"

(Used for testing date, time, and calendar validators.)

Parameters
`java_fmt` (*`str`*) – The Java SimpleDateFormat pattern to convert

Returns
The Python string pattern, that when fed into `datetime.strftime()` produces an equivalent date/time string representation.

`apache_commons_validator_python.util.datetime_helpers.`**`obj_to_str`**(*expected_obj: object*, *tested_obj: object = None*) → str

Prints the object as a string for debugging purposes on the test cases.

Parameters

- **`expected_obj`** (*`Union[datetime, object]`*) – The expected object in the test case.
- **`tested_obj`** (*`Union[datetime, object]`*) – The object being tested in the test case.

Returns
A string comparing the expected_obj and tested_obj and their fields if applicable.

`apache_commons_validator_python.util.datetime_helpers.`**`parse_pattern_flexible`**(*value: str*, *ldml_pattern: str*) → datetime

TODO: Parses value string into the representing datetime in the locale's "short" style. Except there are multiple acceptable "short" strings in Java, but only one acceptable "short" string in Python. This function aims to mimic Java's flexibility. Uses the system's default locale if a locale is not provided.

`apache_commons_validator_python.util.datetime_helpers.`**`parse_pattern_strict`**(*value: str*, *ldml_pattern: str*) → datetime

TODO:

`apache_commons_validator_python.util.datetime_helpers.`**`timezone_gmt`**(*zone: str*) → ZoneInfo

Creates and returns a `tzinfo` object with the specified timezone. A replacement for Java's `org.apache.commons.lang3.time.TimeZones`.

Parameters
`zone` (*`str`*) – The timezone to return. e.g. "est", or "gmt".

Returns

A `ZoneInfo` object with the specified zone. Note that `tzinfo` is an abstract class, and `ZoneInfo` is an implementation.

`apache_commons_validator_python.util.datetime_helpers.`**`timezone_has_same_rules`**`(`*val1: datetime | tzinfo*, *val2: datetime | tzinfo*`)` → bool

Wrapper function for java.util.TimeZone.hasSameRules().

Determines if two datetime or tzinfo objects share the same rules for time adjustments, including the raw UTC offset and daylight saving time rules. This function disregards the time zone identifier (i.e. the name) and focuses solely on the effective behavior (offset) of the time zones.

Parameters

- **`val1`** (*`Union[datetime, tzinfo]`*) – A datetime object (from which the tzinfo is extracted) or a tzinfo instance representing the first time zone.
- **`val2`** (*`Union[datetime, tzinfo, None]`*) – A datetime object or tzinfo instance representing the second time zone. If None, the function returns False.

Returns

`True` if the time zones have identical rules (i.e. the same UTC offset at the reference time); `False` otherwise, or if either value does not have tzinfo information.

apache_commons_validator_python.util.decimal_places module

`apache_commons_validator_python.util.decimal_places.`**`max_decimal_places`**`(`*regex_pattern: str*`)`

apache_commons_validator_python.util.digester module

`class apache_commons_validator_python.util.digester.`**`Digester`**`(`*root_object: ['ValidatorResources']*`)`

Bases: `ContentHandler`

Custom SAX-based XML parser that interprets digester rule files and applies them to dynamically construct and wire objects such as FormSet, Form, Field, etc., based on the XML structure.

This class mimics Apache Commons Digester by using SAX parsing combined with pattern-based rule interpretation, creating a hierarchy of validation resources.

`characters``(`*content: str*`)` → None

Appends character data between start and end tags to the text buffer.

Parameters

`content` (*`str`*) – Character data within an XML tag.

`class_mapping: Dict[str, Type]`

Maps class names (as strings) to their actual Python class types for dynamic instantiation.

`endElement``(`*name: str*`)` → None

Handles logic for end of an XML element during SAX parsing.

Applies call-method-rule if present and wires the current object to its parent via set-next-rule. Also processes call-param-rule for nested values.

Parameters

`name` (*`str`*) – Name of the XML tag.

`load_rules`(*rules_file: str*) → None

Parses the digester rules XML file and loads all patterns and rule bindings.

Parameters
`rules_file` (*str*) – Path to the XML file defining the digester rules.

`parse`(*xml_file: str*) → Any

Parses the input XML file using this digester instance.

Parameters
`xml_file` (*str*) – Path to the XML file to parse.

Returns
The root object with attached parsed structure (usually ValidatorResources).

Return type
Any

`startElement`(*name: str*, *attrs: AttributesImpl*) → None

Handles logic for start of an XML element during SAX parsing.

Applies any factory-create-rule or object-create-rule for the current path, sets properties, and pushes the created object onto the object stack.

Parameters

- `name` (*str*) – Name of the XML tag.
- `attrs` (*AttributesImpl*) – Attributes associated with the tag.

apache_commons_validator_python.util.domains module

Module Name: domains.py

Description: Contains the lists of TLDs from apache.commons.validator.routines.DomainValidator.java Link: https://github.com/apache/commons-validator/blob/master/src/main/java/org/apache/commons/validator/routines/DomainValidator.java

Author: Jessica Breuhaus

License (Taken from apache.commons.validator.routines.DomainValidator.java):

`class apache_commons_validator_python.util.domains.Domains`

Bases: `object`

Contains lists of Infrastructure TLDs, Generic TLDs, Country Code TLDs, and Local TLDs.

`<li>INFRASTRUCTURE_TLDS`

List of Infrastructure TLDs (eg. 'arpa').</li>

Type
list[str]

<li>GENERIC_TLDS

List of Generic TLDs (eg. 'com').</li>

Type
list[str]

<li>COUNTRY_CODE_TLDS

List of Country Code TLDs (eg. 'us').</li>

Type
list[str]

<li>LOCAL_TLDS

List of Local TLDs (eg. 'localhost').</li>

Type
list[str]

```
COUNTRY_CODE_TLDS: Final[list] = ['ac', 'ad', 'ae', 'af', 'ag', 'ai', 'al', 'am',
'ao', 'aq', 'ar', 'as', 'at', 'au', 'aw', 'ax', 'az', 'ba', 'bb', 'bd', 'be', 'bf',
'bg', 'bh', 'bi', 'bj', 'bm', 'bn', 'bo', 'br', 'bs', 'bt', 'bv', 'bw', 'by', 'bz',
'ca', 'cc', 'cd', 'cf', 'cg', 'ch', 'ci', 'ck', 'cl', 'cm', 'cn', 'co', 'cr', 'cu',
'cv', 'cw', 'cx', 'cy', 'cz', 'de', 'dj', 'dk', 'dm', 'do', 'dz', 'ec', 'ee', 'eg',
'er', 'es', 'et', 'eu', 'fi', 'fj', 'fk', 'fm', 'fo', 'fr', 'ga', 'gb', 'gd', 'ge',
'gf', 'gg', 'gh', 'gi', 'gl', 'gm', 'gn', 'gp', 'gq', 'gr', 'gs', 'gt', 'gu', 'gw',
'gy', 'hk', 'hm', 'hn', 'hr', 'ht', 'hu', 'id', 'ie', 'il', 'im', 'in', 'io', 'iq',
'ir', 'is', 'it', 'je', 'jm', 'jo', 'jp', 'ke', 'kg', 'kh', 'ki', 'km', 'kn', 'kp',
'kr', 'kw', 'ky', 'kz', 'la', 'lb', 'lc', 'li', 'lk', 'lr', 'ls', 'lt', 'lu', 'lv',
'ly', 'ma', 'mc', 'md', 'me', 'mg', 'mh', 'mk', 'ml', 'mm', 'mn', 'mo', 'mp', 'mq',
'mr', 'ms', 'mt', 'mu', 'mv', 'mw', 'mx', 'my', 'mz', 'na', 'nc', 'ne', 'nf', 'ng',
'ni', 'nl', 'no', 'np', 'nr', 'nu', 'nz', 'om', 'pa', 'pe', 'pf', 'pg', 'ph', 'pk',
'pl', 'pm', 'pn', 'pr', 'ps', 'pt', 'pw', 'py', 'qa', 're', 'ro', 'rs', 'ru', 'rw',
'sa', 'sb', 'sc', 'sd', 'se', 'sg', 'sh', 'si', 'sj', 'sk', 'sl', 'sm', 'sn', 'so',
'sr', 'ss', 'st', 'su', 'sv', 'sx', 'sy', 'sz', 'tc', 'td', 'tf', 'tg', 'th', 'tj',
'tk', 'tl', 'tm', 'tn', 'to', 'tr', 'tt', 'tv', 'tw', 'tz', 'ua', 'ug', 'uk', 'us',
'uy', 'uz', 'va', 'vc', 've', 'vg', 'vi', 'vn', 'vu', 'wf', 'ws', 'xn--2scrj9c',
'xn--3e0b707e', 'xn--3hcrj9c', 'xn--45br5cyl', 'xn--45brj9c', 'xn--4dbrk0ce',
'xn--54b7fta0cc', 'xn--80ao21a', 'xn--90a3ac', 'xn--90ais',
'xn--clchc0ea0b2g2a9gcd', 'xn--d1alf', 'xn--e1a4c', 'xn--fiqs8s', 'xn--fiqz9s',
'xn--fpcrj9c3d', 'xn--fzc2c9e2c', 'xn--gecrj9c', 'xn--h2breg3eve', 'xn--h2brj9c',
'xn--h2brj9c8c', 'xn--j1amh', 'xn--j6w193g', 'xn--kprw13d', 'xn--kpry57d',
'xn--l1acc', 'xn--lgbbat1ad8j', 'xn--mgb9awbf', 'xn--mgba3a4f16a', 'xn--mgbaam7a8h',
'xn--mgbah1a3hjkrd', 'xn--mgbai9azgqp6j', 'xn--mgbayh7gpa', 'xn--mgbbh1a',
'xn--mgbbh1a71e', 'xn--mgbc0a9azcg', 'xn--mgbcpq6gpa1a', 'xn--mgberp4a5d4ar',
'xn--mgbgu82a', 'xn--mgbpl2fh', 'xn--mgbtx2b', 'xn--mgbx4cd0ab', 'xn--mix891f',
'xn--node', 'xn--o3cw4h', 'xn--ogbpf8fl', 'xn--p1ai', 'xn--pgbs0dh', 'xn--q7ce6a',
'xn--qxa6a', 'xn--qxam', 'xn--rvc1e0am3e', 'xn--s9brj9c', 'xn--wgbh1c',
'xn--wgbl6a', 'xn--xkc2al3hye2a', 'xn--xkc2dl3a5ee0h', 'xn--y9a3aq',
'xn--yfro4i67o', 'xn--ygbi2ammx', 'ye', 'yt', 'za', 'zm', 'zw']
```

```
GENERIC_TLDS: Final[list] = ['aaa', 'aarp', 'abb', 'abbott', 'abbvie', 'abc',
'able', 'abogado', 'abudhabi', 'academy', 'accenture', 'accountant', 'accountants',
'aco', 'actor', 'ads', 'adult', 'aeg', 'aero', 'aetna', 'afl', 'africa', 'agakhan',
'agency', 'aig', 'airbus', 'airforce', 'airtel', 'akdn', 'alibaba', 'alipay',
'allfinanz', 'allstate', 'ally', 'alsace', 'alstom', 'amazon', 'americanexpress',
'americanfamily', 'amex', 'amfam', 'amica', 'amsterdam', 'analytics', 'android',
'anquan', 'anz', 'aol', 'apartments', 'app', 'apple', 'aquarelle', 'arab', 'aramco',
'archi', 'army', 'art', 'arte', 'asda', 'asia', 'associates', 'athleta', 'attorney',
'auction', 'audi', 'audible', 'audio', 'auspost', 'author', 'auto', 'autos', 'aws',
'axa', 'azure', 'baby', 'baidu', 'banamex', 'band', 'bank', 'bar', 'barcelona',
'barclaycard', 'barclays', 'barefoot', 'bargains', 'baseball', 'basketball',
'bauhaus', 'bayern', 'bbc', 'bbt', 'bbva', 'bcg', 'bcn', 'beats', 'beauty', 'beer',
'bentley', 'berlin', 'best', 'bestbuy', 'bet', 'bharti', 'bible', 'bid', 'bike',
'bing', 'bingo', 'bio', 'biz', 'black', 'blackfriday', 'blockbuster', 'blog',
'bloomberg', 'blue', 'bms', 'bmw', 'bnpparibas', 'boats', 'boehringer', 'bofa',
'bom', 'bond', 'boo', 'book', 'booking', 'bosch', 'bostik', 'boston', 'bot',
'boutique', 'box', 'bradesco', 'bridgestone', 'broadway', 'broker', 'brother',
'brussels', 'build', 'builders', 'business', 'buy', 'buzz', 'bzh', 'cab', 'cafe',
'cal', 'call', 'calvinklein', 'cam', 'camera', 'camp', 'canon', 'capetown',
'capital', 'capitalone', 'car', 'caravan', 'cards', 'care', 'career', 'careers',
'cars', 'casa', 'case', 'cash', 'casino', 'cat', 'catering', 'catholic', 'cba',
'cbn', 'cbre', 'center', 'ceo', 'cern', 'cfa', 'cfd', 'chanel', 'channel',
'charity', 'chase', 'chat', 'cheap', 'chintai', 'christmas', 'chrome', 'church',
'cipriani', 'circle', 'cisco', 'citadel', 'citi', 'citic', 'city', 'claims',
'cleaning', 'click', 'clinic', 'clinique', 'clothing', 'cloud', 'club', 'clubmed',
'coach', 'codes', 'coffee', 'college', 'cologne', 'com', 'commbank', 'community',
'company', 'compare', 'computer', 'comsec', 'condos', 'construction', 'consulting',
'contact', 'contractors', 'cooking', 'cool', 'coop', 'corsica', 'country', 'coupon',
'coupons', 'courses', 'cpa', 'credit', 'creditcard', 'creditunion', 'cricket',
'crown', 'crs', 'cruise', 'cruises', 'cuisinella', 'cymru', 'cyou', 'dad', 'dance',
'data', 'date', 'dating', 'datsun', 'day', 'dclk', 'dds', 'deal', 'dealer', 'deals',
'degree', 'delivery', 'dell', 'deloitte', 'delta', 'democrat', 'dental', 'dentist',
'desi', 'design', 'dev', 'dhl', 'diamonds', 'diet', 'digital', 'direct',
'directory', 'discount', 'discover', 'dish', 'diy', 'dnp', 'docs', 'doctor', 'dog',
'domains', 'dot', 'download', 'drive', 'dtv', 'dubai', 'dunlop', 'dupont', 'durban',
'dvag', 'dvr', 'earth', 'eat', 'eco', 'edeka', 'edu', 'education', 'email',
'emerck', 'energy', 'engineer', 'engineering', 'enterprises', 'epson', 'equipment',
'ericsson', 'erni', 'esq', 'estate', 'eurovision', 'eus', 'events', 'exchange',
'expert', 'exposed', 'express', 'extraspace', 'fage', 'fail', 'fairwinds', 'faith',
'family', 'fan', 'fans', 'farm', 'farmers', 'fashion', 'fast', 'fedex', 'feedback',
'ferrari', 'ferrero', 'fidelity', 'fido', 'film', 'final', 'finance', 'financial',
'fire', 'firestone', 'firmdale', 'fish', 'fishing', 'fit', 'fitness', 'flickr',
'flights', 'flir', 'florist', 'flowers', 'fly', 'foo', 'food', 'football', 'ford',
'forex', 'forsale', 'forum', 'foundation', 'fox', 'free', 'fresenius', 'frl',
'frogans', 'frontier', 'ftr', 'fujitsu', 'fun', 'fund', 'furniture', 'futbol',
'fyi', 'gal', 'gallery', 'gallo', 'gallup', 'game', 'games', 'gap', 'garden', 'gay',
'gbiz', 'gdn', 'gea', 'gent', 'genting', 'george', 'ggee', 'gift', 'gifts', 'gives',
'giving', 'glass', 'gle', 'global', 'globo', 'gmail', 'gmbh', 'gmo', 'gmx',
'godaddy', 'gold', 'goldpoint', 'golf', 'goo', 'goodyear', 'goog', 'google', 'gop',
'got', 'gov', 'grainger', 'graphics', 'gratis', 'green', 'gripe', 'grocery',
'group', 'gucci', 'guge', 'guide', 'guitars', 'guru', 'hair', 'hamburg', 'hangout',
'haus', 'hbo', 'hdfc', 'hdfcbank', 'health', 'healthcare', 'help', 'helsinki',
'here', 'hermes', 'hiphop', 'hisamitsu', 'hitachi', 'hiv', 'hkt', 'hockey',
'holdings', 'holiday', 'homedepot', 'homegoods', 'homes', 'homesense', 'honda',
'horse', 'hospital', 'host', 'hosting', 'hot', 'hotels', 'hotmail', 'house', 'how',
'hsbc', 'hughes', 'hyatt', 'hyundai', 'ibm', 'icbc', 'ice', 'icu', 'ieee', 'ifm',
'ikano', 'imamat', 'imdb', 'immo', 'immobilien', 'inc', 'industries', 'infiniti',
'info', 'ing', 'ink', 'institute', 'insurance', 'insure', 'int', 'international',
'intuit', 'investments', 'ipiranga', 'irish', 'ismaili', 'ist', 'istanbul', 'itau',
'itv', 'jaguar', 'java', 'jcb', 'jeep', 'jetzt', 'jewelry', 'jio', 'jll', 'jmp',
```

`INFRASTRUCTURE_TLDS: Final[list] = ['arpa']`

`LOCAL_TLDS: Final[list] = ['localdomain', 'localhost']`

apache_commons_validator_python.util.flags module

class `apache_commons_validator_python.util.flags.`**`Flags`**(*flags=0*)

Bases: `object`

Represents a collection of 64 boolean (on/off) flags. Individual flags are represented by powers of 2. For example, Flag 1 = 1 Flag 2 = 2 Flag 3 = 4 Flag 4 = 8.

or using shift operator to make numbering easier: Flag 1 = 1 << 0 Flag 2 = 1 << 1 Flag 3 = 1 << 2 Flag 4 = 1 << 3

There cannot be a flag with a value of 3 because that represents Flag 1 and Flag 2 both being on/true.

`clear`()

Turns off all flags.

This is a synonym for *turnOffAll()*

property flags

Returns the current flags.

Returns
collection of boolean flags represented.

`is_off`(*flag*)

Tests whether the given flag is off. If the flag is not a power of 2 (for example, 3) this tests whether the combination of flags is off.

Parameters
flag – Flag value to check.

Returns
whether the specified flag value is off.

`is_on`(*flag*)

Tests whether the given flag is on.

If the flag is not a power of 2 for example, 3) this tests whether the combination of flags is on.

Parameters
flag – Flag value to check.

Returns
whether the specified flag value is on.

turn_off(*flag*)

Turns off the given flag. If the flag is not a power of 2 (for example, 3) this turns off multiple flags.

Args

flag: Flag value to turn off.

turn_off_all()

Turn off all flags.

turn_on(*flag*)

Turns on the given flag. If the flag is not a power of 2 (for example, 3) this turns on multiple flags.

Parameters

flag – Flag value to turn on.

turn_on_all()

Turn on all 64 flags.

apache_commons_validator_python.util.locale module

class `apache_commons_validator_python.util.locale.`**Locale**(*locale_str=None*, *language=None*, *country=''*, *variant=''*)

Bases: `object`

Python wrapper of locale that implements some java.util.Locale functionality with getters and setters.

COUNTRY_NAMES = {'CN': 'China', 'DE': 'Germany', 'ES': 'Spain', 'FR': 'France', 'JP': 'Japan', 'US': 'United States'}

ISO3_COUNTRY_MAP = {'CN': 'CHN', 'DE': 'DEU', 'ES': 'ESP', 'FR': 'FRA', 'JP': 'JPN', 'US': 'USA'}

ISO3_LANGUAGE_MAP = {'de': 'deu', 'en': 'eng', 'es': 'spa', 'fr': 'fra', 'ja': 'jpn', 'zh': 'zho'}

LANGUAGE_NAMES = {'de': 'German', 'en': 'English', 'es': 'Spanish', 'fr': 'French', 'ja': 'Japanese', 'zh': 'Chinese'}

property country

Returns the country code (e.g., 'US')

property display_country

Returns a human-readable country name.

property display_language

Returns a human-readable language name.

property display_variant

Returns a human-readable variant (if applicable)

classmethod getdefaultlocale()

Mimics Java's Locale.getDefault()

property iso3_country

Converts 2-letter country code to ISO 3166-1 alpha-3 code.

property iso3_language

Converts 2-letter language code to ISO 639-2 (3-letter) code.

property language

Returns the language code (e.g., 'en')

property variant

Returns the variant code (e.g., 'POSIX') if available.

apache_commons_validator_python.util.regex module

Module Name: re.py Description:

Provides a wrapper class for python's `re` module, by overriding some of the methods so their specifications more closely match the corresponding methods in Java's `Pattern` package.

Used internally in *src/main/routines/regex_validator.py*, and dependencies.

Author: Juji Lau

Substitutions:

Java uses the package `Pattern` for regular expressions. This Python file uses the `re` package.

Substitutions (Java -> Python):

`java.util.regex` -> `re` Regex package `regex.compile()` -> `re.compile()` Compiles a Pattern `regex.Pattern` -> `re.Pattern` `Pattern.CASE_INSENSITIVE` -> `re.INGORECASE` Flag to ignore case when pattern matching. `Pattern.pattern` -> `Pattern.pattern` Field that represents the pattern regex as a string. `Pattern.matcher(value).matches()` -> `Pattern.fullmatch(value)` Matches the entire value against the pattern.

Java:

`Pattern.matcher(value)` Creates a `Matcher` object that matches the entire value against the pattern. `matches()` Returns `True` iff the entire value matches the regex pattern.

Python:

`Pattern.fullmatch()` Creates a `Match` object that matches the entire value against the pattern. None if there is no match.

`regex.Matcher` -> `re.Match` Object created by calling method(s) on `Pattern`. `Matcher.groups()` -> `Match.groups()` List of all the matches in the string to the pattern regex. `java.lang.Object.clone()` -> `copy.copy()` For shallow copies

class `apache_commons_validator_python.util.regex.`**`Regex`**

Bases: `object`

A partial wrapper class for Python's `re` module, emulating specific functionalities in Java's `Pattern` package.

This class includes only the methods and attributes pertinent to the translation project, omitting other functionalities of Java's `Pattern` class.

CASE_INSENSITIVE

Flag to perform case-insensitive matching, equivalent to `re.IGNORECASE`.

Type

int

CASE_INSENSITIVE: int = 2

`static compile`(*pattern_str: str*, *flags: int | None = 0*) → Pattern

Compile a regular expression pattern into a Pattern object. This method emulates Java's `Pattern.compile()` method.

Parameters

- **`pattern`** (*`str`*) – The regular expression pattern to compile.
- **`flags`** (*`int, optional`*) – Flags to modify the regular expression's behavior. Defaults to 0.

Returns
The compiled regular expression pattern object.

Return type
Pattern

`classmethod pattern_matches`(*pattern: Pattern*, *string: str*) → bool

Determines if the entire string matches the given pattern.

This method serves as a substitute for Java's `Pattern.matcher().matches()`, providing equivalent functionality in Python.

Parameters

- **`pattern`** (*`Pattern`*) – The compiled regular expression pattern.
- **`string`** (*`str`*) – The string to be matched against the pattern.

Returns
True if the entire string matches the pattern; False otherwise.

Return type
bool

apache_commons_validator_python.util.validator_utils module

`class apache_commons_validator_python.util.validator_utils.ValidatorUtils`

Bases: `object`

`copy_map`() → dict[str, object]

Makes and returns a deep copy of a map if the values are Msg, Arg, or Var, and a shallow copy otherwise.

Parameters
`map` (*`dict[str, object]`*) – The input map to copy

Returns
The copied map, where for each entry, a deepcopy is made if the value is an Arg, Var, or Msg, and a shallow copy otherwise.

`get_value_as_string`(*property: str*) → str

Returns the value from the bean property as a string.

Parameters

- **`bean`** (*`object`*) – An instance of a class
- **`property`** (*`str`*) – A field in bean

Returns

- "" If property is an empty list, a list of empty strings, or an empty Collection

- The result of property.toStr()
- None if there's an error.

`classmethod` **`replace`**(*value: str*, *key: str*, *replace_value: str*) → str

Replaces a key part of value with replaceValue.

Parameters

- **value** (*str*) – The string to perform the replacement on
- **key** (*str*) – The name of the constant
- **replace_value** (*str*) – The value of hte constant

Returns

The modified value.

`apache_commons_validator_python.util.validator_utils.`**`integer_compare`**(*a: int*, *b: int*) → int

Compares a, and b.

Parameters

- **a** (*int*) – The first value to compare
- **b** (*int*) – The second value to compare

Returns

0 if a == b. -1 if a < b. 1 if a > b.

`apache_commons_validator_python.util.validator_utils.`**`to_lower`**(*s: str*) → str | None

Returns s with all letters lowercased, and leading and trailing whitespaces removed.

Parameters

s (*str*) – The string to process

Returns

s with the leading and trailing whitespaces removed, and all letters lowercased. None if s is an invalid argument.

Module contents

1.1.2 Submodules

1.1.3 apache_commons_validator_python.arg_new module

`class apache_commons_validator_python.arg_new.`**`Arg`**

Bases: `object`

A default argument or an argument for a specific validator definition (ex: required) can be stored to pass into a message as parameters. This can be used in a pluggable validator for constructing locale sensitive messages by using *MessageFormat* or an equivalent class. The resource field can be used to determine if the value stored in the argument is a value to be retrieved from a locale sensitive message retrieval system like `java.util.PropertyResourceBundle`. The resource field defaults to 'true'.

Instances of this class are configured with an <arg> xml element.

Taken from apache.commons.validator.Arg;

`serializable`

Indicates if the object is serializable.

Type
bool

`cloneable`

Indicates if the object can be cloned.

Type
bool

`bundle`

The resource bundle name that this Arg's *key* should be resolved in (optional).

Type
str

`key`

The key or value of the argument.

Type
str

`name`

The name dependency that this argument goes with (optional).

Type
str

`position`

This argument's position in the message. Set position=0 to make a replacement in this string: "some msg {0}". @since 1.1

Type
int

`resource`

Whether or not the key is a message resource (optional). Defaults to True. If it is 'true', the value will try to be resolved as a message resource.

Type
bool

`property bundle: str | None`

Get the resource bundle name.

Returns
bundle (str)

`cloneable = True`

property key: str | None

Gets the key/value.

Returns
key (str)

property name: str | None

Gets the name of the dependency.

Returns
name (str)

property position: int

Gets the replacement position.

Returns
position (int)

property resource: bool

Tests whether or not the key is a resource key or literal value.

Returns
resource (bool)

`serializable = True`

1.1.4 apache_commons_validator_python.field_new module

class apache_commons_validator_python.field_new.**Field**

Bases: `object`

This contains the list of pluggable validators to run on a field and any message information and variables to perform the validations and generate error messages. Instances of this class are configured with a <field> xml element.

Taken from apache.commons.validator.Field;

serializable

Indicates if the object is serializable.

Type
bool

cloneable

Indicates if the object can be cloned.

Type
bool

`TOKEN_INDEXED: Final[str] = '[]'`

`add_arg`(*arg:* Arg) → None

Add an Arg to the replacement argument list.

Parameters
`arg` (`Arg`) – Validation message's argument.

`add_msg`(*msg:* Msg) → None

Add a Msg to the Field.

Parameters
`msg` (`Msg`) – A validation message.

`add_var`(*arg0*, *arg1=None*, *arg2=None*) → None

Add a Var to the Field.

Example Method Call 1:
v = Var(name, value, js_type) add_var(arg0=v)

Example Method Call 2:
add_var(arg0=name, arg1=value, arg2=jsType)

Parameters

- **`arg0`** (*`str | Var`*) – Either the Validator Argument or the name of the validation.
- **`arg1`** (*`str | None`*) – The Argument's value
- **`arg2`** (*`str | None`*) – The JavaScript type

Returns
None

`property client_validation: bool`

Determines whether client-side scripting should be generated for this field. The default is true.

Returns true for scripting; otherwise false.

`clone`() → *Field*

Creates and returns a copy of this field.

Returns
A copy of the Field.

`cloneable = True`

`property dependency_list: List[str]`

Gets an copy (so it is unmodifiable) list of the dependencies in the same order they were defined in the parameter passed to depends.

Returns
A lit of the Field's dependencies.

Return type
dependency_list (List[*Field*])

property depends: str | None

Gets the validation rules for this field as a comma separated list. (translation of getDepends())

Returns
A comma separated list of validator names.

property field_order: int | None

Gets the position of the {@code Field} in the validation list. (translation of getFieldOrder())

Returns
the field position

Return type
field_order (Optional[int])

property field_property: str | None

Gets the property name of the field.

generate_key() → None

Generates correct key value.

Returns
None

get_arg(*position: int*, *key: str | None*) → *Arg* | None

Gets the Arg object at the given position. If the key finds a None value, then the default value then the default value will be retrieved.

Parameters

- **position** (*int*) – The Arg number to find.
- **key** (*str | None*) – The name the Arg is stored under. If not found, the default Arg for the given position (if any) will be retrieved.

Returns
The Arg with the given name and position or None if not found.

get_args(*key: str*) → List[*Arg* | None]

Retrieves the Args for the given validator name.

Parameters
key (*str*) – The validator's args to retrieve.

Returns
A List of Args sorted by the Args' positions.

get_message(*key: str*) → *Msg* | None

Retrieve a message object.

Parameters
key (*str*) – Validation key

Returns
A validation message for a specified validator.

get_msg(*key: str*) → str | None

Retrieve a message value.

Parameters
key (*str*) – Validation key

Returns
A validation message for a specified validator.

`get_msg_map()` → Dict[str, *Msg*]

Returns a Dict of String Msg names to Msg objects.

`get_var(`*main_key: str*`)` → *Var* | None

Retrieve a variable.

Parameters
main_key (*str*) – the Variable's key

Returns
the Variable

`get_var_map()` → Dict[str, *Var*]

Returns a Map of String Var names to Var objects.

`get_var_value(`*main_key: str*`)` → str | None

Retrieve a variable's value.

Parameters
main_key (*str*) – the Variable's key

Returns
the Variable's value

`property indexed_list_property: str | None`

Gets the indexed property name of the field. (translation of getIndexedListProperty())

Returns
the field's indexed list property name.

Return type
indexed_list_property (Optional[str])

`property indexed_property: str | None`

Gets the indexed property name of the field.

`property indexed_property_bean: List[Any]`

Returns an indexed property from the object we're validating.

Parameters
bean (*object*) – The bean to extract the indexed values from.

Raises: ValidatorException If there's an error looking up the property
or, the property found is not indexed.

`is_dependency(`*validator_name: str*`)` → bool

Checks if the validator is listed as a dependency.

Parameters
validator_name (*str*) – the name of the validator to check

Returns: whether the field is dependent on a validator

`is_indexed()` → bool

If there is a value specified for the __indexed_property field then true will be returned. otherwise it will be false.

Returns: whether the Field is indexed.

`property key: str`

Gets a unique key based on the property and indexedProperty fields. (translation of getKey())

Returns
a unique key for the field.

Return type
key (str)

`property msgs: Dict[str, Msg]`

The Field's messages are returned as a copied dictionary so it doesn't modify the original.

Returns
Dict[str, "Msg"] of validation messages for the field.

`property page: int | None`

Gets the page value that the Field is associated with for validation.

`process`(*global_constants: Dict[str, str]*, *constants: Dict[str, str]*) → None

Replace the constants with values in fields and process the depends field to create the dependency dict.

Parameters

- **global_constants** (*Dict[str, str]*)
- **constants** (*Dict[str, str]*)

`serializable = True`

`validate`(*params: Dict[str, object]*, *actions: Dict[str,* ValidatorAction*]*) → *ValidatorResults*

Run the configured validations on this field. Run all validations in the depends clause over each item in turn, returning when the first one fails.

Parameters

- **params** (*Dict[str, object]*) – A dict of parameter class names to parameter values to pass into validation methods.
- **objects.** (*actions A dict of validator names to ValidatorAction*)

Returns
A ValidatorResults object containing validation messages for this field.

Throws:
ValidatorException If an error occurs during validation.

`property vars: Dict[str, Var]`

The Field's variables are returned as a copied dictionary so it doesn't modify the original.

Returns
Dict[str, "Msg"] of Variable's for a field.

1.1.5 apache_commons_validator_python.form_new module

`class apache_commons_validator_python.form_new.`**`Form`**

Bases: `object`

This contains a set of validation rules for a form. The information is contained in a list of *Field* objects. Instances of this class are configured with a <form> xml element.

Taken from apache.commons.validator.Form;

`add_field`(*field:* Field) → None

Add a *Field* to the *Form*.

Parameters
`field` (`Field`) – the field

`cloneable = False`

`contains_field`(*field_name: str*) → bool

Returns true if this Form contains a Field with the given name.

Parameters
`field_name` (*`str`*) – the field name

Returns
True if this form contains the field by the given name.

`property fields: List[`*`Field`*`]`

A copy of list of `Field`s is returned.

`get_extends`() → str

Gets the name/key of the parent set of validation rules.

Returns
the extends value

`get_field`(*field_name: str*) → *Field* | None

Returns the Field with the given name or None if this Form has no such field.

Parameters
`field_name` (*`str`*) – the field name

Returns
the field value

`get_field_map`() → Dict[str, *Field*]

Returns a dict of str field keys to Field objects.

Returns
the field map value

`is_extending()` → bool

Gets extends flag.

`property name: str`

Gets the name of the form.

Returns
The name of the form.

`property processed: bool`

Checks whether the form has been processed.

(translation of isProcessed())

Returns
True if the form has been processed, False otherwise.

`serializable = True`

`set_extends`(*inherit: str*) → None

Sets the name/key of the parent set of validation rules.

Parameters
`value` (*`the new extends`*)

`validate`(*params: dict*, *actions: dict*, *page: int*, *field_name: str = None*) → *ValidatorResults*

Validates the fields of the form and returns the validation results.

This method iterates through the form's fields and validates them based on the provided *params*, *actions*, and *page* parameters. If a *field_name* is provided, it validates only that specific field. Otherwise, it validates all fields in the form that are relevant for the given page.

Parameters

- **`params`** (*`dict`*) – A dictionary containing parameters required for validation.
- **`actions`** (*`dict`*) – A dictionary of actions associated with the validation process.
- **`page`** (*`int`*) – The current page number used for validating fields relevant to this page.
- **`field_name`** (*`str, optional`*) – The specific field to validate. If not provided, all fields on the current page are validated.

Returns
An object containing the result of the validation process.

Return type
ValidatorResults

Raises
`ValidatorException` – If the specified *field_name* does not correspond to a valid field in the form.

1.1.6 apache_commons_validator_python.form_set_factory_new module

`class apache_commons_validator_python.form_set_factory_new.`**`FormSetFactory`**

Bases: `object`

Factory class used to create FormSet instances.

`create_form_set`(*resources: ValidatorResources*, *language: str | None*, *country: str | None*, *variant: str | None*) → *FormSet*

Creates or retrieves a FormSet based on the locale attributes.

Parameters

- **resources** (`ValidatorResources`) – The validator resources containing form sets.
- **language** (*Optional[str]*) – The locale's language.
- **country** (*Optional[str]*) – The locale's country.
- **variant** (*Optional[str]*) – The locale's variant.

Returns

The FormSet instance for the given locale.

Return type

FormSet

`create_object`(*attributes*, *resources: ValidatorResources*) → *FormSet*

Creates or retrieves a FormSet based on XML attributes.

Parameters

- **attributes** (*Attributes*) – The SAX attributes for the FormSet element.
- **resources** (`ValidatorResources`) – The validator resources.

Returns

The created or retrieved FormSet instance.

Return type

FormSet

1.1.7 apache_commons_validator_python.form_set_new module

`class apache_commons_validator_python.form_set_new.FormSet`

Bases: `object`

This class contains a set of Forms associated with a specific Locale. It supports operations for managing Forms, Constants, and Locale components (language, country, variant). It also provides methods for processing Forms, merging FormSets, and managing their states.

`serializable`

Indicates if the object is serializable.

Type
bool

`cloneable`

Indicates if the object can be cloned.

Type
bool

`language`

The language component of the Locale.

Type
Optional[str]

`country`

The country component of the Locale.

Type
Optional[str]

`variant`

The variant component of the Locale.

Type
Optional[str]

`processed`

Indicates if the FormSet has been processed.

Type
bool

`merged`

Indicates if the FormSet has been merged with a parent.

Type
bool

forms

A dictionary of Forms in the FormSet.

Type
Dict[str, 'Form']

constants

A dictionary of Constants in the FormSet.

Type
Dict[str, str]

add_constant(*name: str*, *value: str*) → None

Adds a Constant to the FormSet.

Parameters

- **name** (*str*) – The constant name.
- **value** (*str*) – The constant value.

add_form(*f:* Form) → None

Adds a Form to the FormSet.

Parameters
f (Form) – The Form to be added.

cloneable = False

property country: str | None

Returns the country component of the Locale.

display_key() → str

Returns a string representation of the FormSet key based on its Locale components.

Returns
A string representation of the key.

Return type
str

get_form(*form_name: str*) → *Form* | None

Retrieves a Form from the FormSet by its name.

Parameters
form_name (*str*) – The name of the form to retrieve.

Returns
The requested Form, or None if not found.

Return type
Form

get_forms() → Dict[str, *Form*]

A dict of forms is returned as an unmodifiable dict with the key based on the for name.

(translation of getForms())

property language: str | None

Returns the language component of the Locale.

property merged: bool

Returns whether the FormSet has been merged.

process(*global_constants: Dict[str, str]*) → None

Processes all Forms in the FormSet.

Parameters

global_constants (*Dict[str, str]*) – Global constants to be used during processing.

property processed: bool

Returns whether the FormSet has been processed.

serializable = True

property variant: str | None

Returns the variant component of the Locale.

1.1.8 apache_commons_validator_python.generic_type_validator_new module

class apache_commons_validator_python.generic_type_validator_new.**GenericTypeValidator**

Bases: `object`

GenericTypeValidator class provides methods to format and validate different types of data inputs.

Use NumPy data types for byte, double, short, and long from Java version since these do not exist in Python.

cloneable: Final[bool] = False

static format_byte(*value: str | None*) → int | None

Method to convert a string value to an integer (byte)

Parameters

value (*str*) – the value validation is being performed on

Returns: the converted integer (byte) value

static format_byte_locale(*value: str | None*, *locale: str | None = None*) → int | None

Method to convert a string value to an integer (byte) with optional locale support.

Parameters

- **value** (*str*) – the value validation is being performed on
- **locale** (*str) the locale to use to parse the number (system default if null*)

Returns: the converter integer (byte) value `static format_credit_card`(*value: str | None*) → int | None

Method to check if a string value represents a valid credit card and convert it to an integer.

Parameters
`value` (*str*) – the valie validation is being performed on

Returns: the converted Credit Card number

`static format_date`(*value: str | None*, *locale: str | None = None*) → datetime | None

Method to convert a string value to a datetime object (date) using the system's locale.

Parameters

- `value` (*str*) – the value validation is being performed on
- `locale` (*str*) – the locale to use to parse the data (system default if null)

Returns: the converted Date value

`static format_date_pattern`(*value: str | None*, *date_pattern: str | None*, *strict: bool*) → datetime | None

Method to convert a string value to a datetime object using a custom date pattern.

Parameters

- `value` (*str*) – the value validation is being performed on
- `date_pattern` (*str*) – the pattern
- `strict` (*bool*) – whether or not to have an exact match of the date_pattern

Returns: the converted Date value

`static format_double`(*value: str | None*) → float | None

Method to convert a string value to a float with optional locale support.

Parameters
`value` (*str*) – the value validation is being performed on

Returns: the converted Double value

`static format_double_locale`(*value: str | None*, *locale: str | None = None*) → float | None

Format a string value into a float (double-precision), respecting the provided locale.

Returns
None if the value is invalid.

`static format_float`(*value: str | None*) → float | None

Method to convert a string value to a float.

Parameters
`value` (*str*) – the value validation is being performed on

Returns: the converted Float value

`static format_float_locale`(*value: str | None*, *locale: str | None = None*) → float | None

Method to convert a string value to a float with locale support.

Parameters

- `value` (*str*) – the value validation is being performed on
- `locale` (*str*) – the locale to use to parse the number (system default if None)

Returns: the converted float value

static format_int(*value: str | None*) → int | None

Checks if the value can safely be converted to an int primitive.

Parameters
value (*str*) – the value validation is being performed on

Returns: the converted int value

static format_int_locale(*value: str | None*, *locale: str | None = None*) → int | None

Format a string value into an integer, respecting the provided locale.

Returns None if the value is invalid or out of range.

static format_long(*value: str | None*) → int | None

Format a string value into a long integer (64-bit).

Returns None if the value is invalid or out of range.

static format_long_locale(*value: str | None*, *locale: str | None = None*) → int | None

Format a string value into a long integer (64-bit), respecting the provided locale.

Returns None if the value is invalid or out of range.

static format_short(*value: str | None*) → int | None

Format a string value into a short integer (16-bit).

Returns None if the value is invalid or out of range.

static format_short_locale(*value: str | None*, *locale: str | None = None*) → int | None

Format a string value into a short integer (16-bit), respecting the provided locale.

Returns None if the value is invalid or out of range.

serializable: Final[bool] = True

1.1.9 apache_commons_validator_python.generic_validator_new module

class apache_commons_validator_python.generic_validator_new.**GenericValidator**

Bases: `object`

This class contains basic methods for performing validations.

Removed functions for double, long, and short for is_in_range (just have one general is_in_range).

static is_blank_or_null(*value: str*) → bool

Checks if the field isn't null and the length of the field is greater than zero, not including whitespace.

1.1.10 apache_commons_validator_python.msg_new module

`class apache_commons_validator_python.msg_new.`**`Msg`**

Bases: `object`

The Msg class represents a message that can be associated with a *Field* and a pluggable validator.

It allows customization of the message, enabling alternative messages to be used instead of the default stored in the *ValidatorAction*. Instances are configured with a <msg> XML element.

`property bundle: str | None`

Gets the resource bundle name. (translatin of getBundle())

Returns
bundle (str)

`clone()` → *Msg*

Creates and returns a deep copy of the current Msg instance using serialization. This method utilizes *pickle* to serialize and deserialize the object.

# TODO: technically this should be overriding __copy__ and __deepcopy__

`cloneable: Final[bool] = True`

`property key: str | None`

Gets the key value. (translation of getKey())

Returns
key (str)

`property name: str | None`

Gets the dependency name. (translation of getName())

Returns
name (str)

`property resource: bool`

Tests whether the key is a resource key or a literal value. (translation of isResource())

Returns
resource (bool)

`serializable: Final[bool] = True`

`exception apache_commons_validator_python.msg_new.`**`UnsupportedOperationException`**

Bases: `Exception`

Custom exception raised when an unsupported operation is attempted.

1.1.11 apache_commons_validator_python.validator_action_new module

class `apache_commons_validator_python.validator_action_new.`**`ValidatorAction`**

Bases: `object`

Contains the information to dynamically create and run a validation method. This is the class representation of a pluggable validator that can be defined in an xml file with the <validator> element.

`property class_name: str`

Returns the class name of the class containing the validation method.

Returns
class_name (str)

`property depends: str`

Gets the other *ValidatorAction`s* that this one depends on.

Returns
depends (str)

`execute_validation_method`(*validator*, *params*)

Executes the validation method.

Parameters

- **`validator`** – Validator instance (context).
- **`params`** – Validation parameters.

Returns
Result of the validation (e.g., True/False).

`getJavascript`()

Gets the name to be used if javascript is generated.

`get_dependencies`() → List[str]

Returns dependencies as a list.

`init`()

Dynamically load the validator class/module.

`property method: str`

Returns the method name.

Returns
method_name (str)

`property name: str`

Returns the name fo the validation.

Returns
name (str)

`setJavascript`(*js_function*) → str | None

Sets the field to contain the name to be used if JavaScript is generated.

Parameters
`js_function` (*`str`*) – the name to be used if JavaScript is generated

1.1.12 apache_commons_validator_python.validator_exception_new module

exception `apache_commons_validator_python.validator_exception_new.`**`ValidatorException`**`(`*message=None*`)`

Bases: `Exception`

The base exception for the Validator Framework. All other *Exception`s thrown during calls to `Validator.validate()* are considered errors.

Taken from org.apache.commons.validator.ValidatorException;

`cloneable = False`

is the class cloneable

`serializable = True`

is the class serializable

1.1.13 apache_commons_validator_python.validator_new module

class `apache_commons_validator_python.validator_new.`**`Validator`**`(`*resources:* ValidatorResources, *form_key: str*, *parameters: Dict[str, Any] | None = None*`)`

Bases: `object`

Core class responsible for validating JavaBeans against a set of validation rules.

Equivalent to org.apache.commons.validator.Validator in the Java version.

`FIELD_PARAM: str = 'field'`

`VALIDATOR_RESULTS_PARAM: str = 'ValidatorResults'`

`get_form()` → *Form* | None

Returns form to validate.

`get_parameter``(`*key: str*`)` → Any

Get value for key in parameters.

Parameters
key (*str*)

Returns
value (Any)

`get_result()` → *ValidatorResults*

Get ValidatorResults of validating the form

Raises
`ValidatorException` –

Returns
ValidatorResults

set_locale(*language: str*, *country: str = None*, *variant: str = None*)

Set the locale of the validator.

Parameters

- **language** (*str*)
- **country** (*str, optional*)
- **variant** (*str, optional*)

set_only_return_errors(*only_errors: bool*) → None

Sets only_return_errors

Parameters
only_errors (*bool*) – if true only return fields that don't pass validation.

set_page(*page: int*) → None

Set the current page number to validate

Parameters
page (*int*)

set_parameter(*key: str*, *value: Any*) → None

Add key, value to parameters.

Parameters

- **key** (*str*)
- **value** (*Any*)

validate_field(*field_name: str*) → *ValidatorResults*

Validate the field in the form.

Parameters
field_name (*str*) – field name in form to validate.

Raises
ValidatorException –

Returns
ValidatorResults

1.1.14 apache_commons_validator_python.validator_resources_new module

class apache_commons_validator_python.validator_resources_new.**ValidatorResources**(*sources: List[str] | None = None*)

Bases: `object`

General purpose class for storing FormSet objects based on their associated locale.

The xml files being passed in need to match the structure detailed below: <form-validation> ├── <global> │ ├── <validator> │ │ ├── name (attribute, required) │ │ ├── classname (attribute, required) │ │ ├── method (attribute, optional, defaults to 'validate') │ │ ├── methodParams (attribute, optional) │ │ ├── depends (attribute, optional) │ │ └── msg (attribute, optional; default error message key) │ ├── <constant> │ │ ├── <constant-name> (text, required) │ │ └── <constant-value> (text, required) │ └── … (multiple <validator> and <constant> allowed) │ ├── <formset> │ ├── language (attribute, optional) │ ├── country (attribute, optional) │ └── <form> │ ├── name (attribute, required) │ └── <field> │ ├── property (attribute, required) │ ├── depends (attribute, optional; comma-separated validator names) │ ├── page (attribute, optional; for multi-page forms) │ ├── indexedListProperty (attribute, optional) │ ├── indexedProperty (attribute, optional) │ ├── key (attribute, optional; alternate for message bundle) │ ├── <arg0> to <arg3> (optional; provides values for messages) │ │ ├── key (attribute, required) │ │ ├── name (attribute, optional) │ │ └── resource (attribute, optional, default true) │ ├── <msg> (optional; overrides default validator message) │ │ ├── name (attribute, required; matches a validator) │ │ └── key (attribute, required; message key) │ ├── <var> (optional; parameter to validator) │ │ ├── <var-name> (text, required) │ │ └── <var-value> (text, required) │ └── … (multiple <arg>, <msg>, <var> allowed) │ └── … (multiple <formset> allowed, one per locale if needed)

add_constant(*name: str*, *value: str*) → None

Add a global constant to the resource.

Parameters

- **name** (*str*) – name of the global constant
- **value** (*str*) – value of the global constant

add_form_set(*form_set:* FormSet) → None

Add a FormSet to this ValidatorResources object.

Parameters
form_set (`FormSet`) – FormSet to add

add_validator_action(*validator_action:* ValidatorAction) → None

Add a ValidatorAction to the resource.

Parameters
validator_action (`ValidatorAction`) – ValidatorAction to add to the resource.

build_locale(*lang: str*, *country: str*, *variant: str*) → str

Assembles a locale code from given parts.

Parameters

- **lang** (*str*)
- **country** (*str*)
- **variant** (*str*)

Returns
str

cloneable = False

get_form(**args*) → *Form*

Gets a Form based on either on language, country, variant and formkey or locale and form key.

Raises
ValueError –

Returns
form (Form)

get_validator_action(*key: str*) → 'ValidatorAction' | None

Gets the ValidatorAction associated with the key.

Returns
ValidatorAction | None

get_validator_actions() → Dict[str, 'ValidatorAction']

Returns a copy of the ValidatorActions in this resources.

process()

Processes the ValidatorResources object.

serializable = True

1.1.15 apache_commons_validator_python.validator_result_new module

class apache_commons_validator_python.validator_result_new.**ValidatorResult**(*field*)

Bases: `object`

Contains the results of a set of validation rules processed on a JavaBean.

class ResultStatus(*valid: bool*, *result: object = None*)

Bases: `object`

Contains the status of a validation.

cloneable = False

property result: object

Gets the result returned by a validation method.

serializable = True

property valid: bool

Returns whether or not the validation passed.

add(*validator_name: str*, *result: bool*, *value: object = None*) → None

Add the result of a validator action.

Parameters

- **validator_name** (*str*) – Name of the validator.
- **result** (*bool*) – Whether the validation passed.
- **value** (*object, optional*) – Value returned by the validator.

cloneable = False

`contains_action`(*validator_name: str*) → bool

Indicates whether a specified validator is in the result.

Parameters
`validator_name` (*str*) – Name of the validator.

Returns
True if the validator is in the result; False otherwise.

Return type
bool

`property field`

The field that was validated.

`get_action_map`() → MappingProxyType

Gets an unmodifiable mapping of validator actions.

Returns
A read-only dictionary mapping validator names to ResultStatus objects.

Return type
MappingProxyType

`get_actions`() → Iterator[str]

Gets an iterator of the action names contained in this result.

Returns
An iterator over the validator action names.

Return type
Iterator[str]

`get_result`(*validator_name: str*)

Gets the result of a validation.

Parameters
`validator_name` (*str*) – Name of the validator.

Returns
The result returned by the validator, or None if not found.

Return type
object

`is_valid`(*validator_name: str*) → bool

Indicates whether a specified validation passed.

Parameters
`validator_name` (*str*) – Name of the validator.

Returns
True if the validation passed; False otherwise.

Return type
bool

`serializable = True`

1.1.16 apache_commons_validator_python.validator_results_new module

class `apache_commons_validator_python.validator_results_new.`**`ValidatorResults`**

Bases: `object`

Contains the results of a set of validation rules processed on a JavaBean.

`add`(*field: Field*, *validator_name: str*, *result: bool*, *value: Any | None = None*) → None

Add the result of a validator action.

Parameters

- **`field`** (`Field`) – The field that was validated.
- **`validator_name`** (*`str`*) – The name of the validator.
- **`result`** (*`bool`*) – The result of the validation.
- **`value`** (*`Any, optional`*) – The value returned by the validator.

`clear`()

Clear all results recorded by this object.

`get_action_map`(*key: str*) → MappingProxyType | None

Gets an unmodifiable mapping of validator actions for a specific field key.

Parameters

`key` (*`str`*) – The key generated from Field.

Returns

A read-only dictionary mapping validator names to ResultStatus objects.

Return type

MappingProxyType

`get_property_names`() → Set[str]

Gets the set of property names for which at least one message has been recorded.

Returns

An unmodifiable set of the property names.

Return type

Set[str]

`get_result_value_map`() → Dict[str, Any]

Gets a map of any objects returned from validation routines.

Returns

Map of objects returned by validators.

Return type

Dict[str, Any]

get_validator_result(*key: str*) → *ValidatorResult* | None

Gets the ValidatorResult associated with the key.

Parameters
`key` (*str*) – The key generated from Field (often just the field name).

Returns
The result of a specified key.

Return type
ValidatorResult

is_empty() → bool

Gets true if there are no messages recorded in this collection.

Returns
Whether these results are empty.

Return type
bool

merge(*other:* ValidatorResults)

Merge another ValidatorResults into this one.

Parameters
`other` (`ValidatorResults`) – ValidatorResults to merge.

1.1.17 apache_commons_validator_python.var_new module

class apache_commons_validator_python.var_new.**Var**(*name=None*, *value=None*, *js_type=None*)

Bases: `object`

JSTYPE_INT: Final[str] = 'int'

JSTYPE_REGEXP: Final[str] = 'regexp'

JSTYPE_STRING: Final[str] = 'string'

property bundle

Get the resource bundle name for the variable (used when resource is True).

clone()

Create and return a shallow copy of this Var instance.

Returns
A clone of the current instance.

Return type
Var

`cloneable = True`

`property js_type`

Get the JavaScript type of the variable.

`property name`

Get the name of the variable.

`property resource`

Indicates whether the value is a resource (True) or a literal value (False).

`serializable = True`

`property value`

Get the value of the variable.

1.1.18 Module contents

PYTHON MODULE INDEX

INDEX