Frappe Utile

أتراب أحمد الدبعي

Utility Functions

Frappe Framework comes with various utility functions to handle common operations for managing site-specific DateTime management, date and currency formatting, PDF generation, and much more.

These utility methods can be imported from the `frappe.utils` module (and its nested modules like `frappe.utils.logger` and `frappe.utils.data`) in any Python file of your Frappe App. This list is not at all exhaustive, you can take a peek at the Framework codebase to see what's available.

now_

`now()`

Returns the current datetime in the format **yyyy-mm-dd hh:mm:ss**

```
from frappe.utils import now

now() # '2021-05-25 06:38:52.242515'
```

getdate_

`getdate(string_date=None)`

Converts `string_date` (yyyy-mm-dd) to `datetime.date` object. If no input is provided, current date is returned. Throws an exception if `string_date` is an invalid date string.

```
from frappe.utils import getdate

getdate() # datetime.date(2021, 5, 25)
getdate('2000-03-18') # datetime.date(2000, 3, 18)
```

today_

`today()`

Returns current date in the format **yyyy-mm-dd**.

```
from frappe.utils import today

today() # '2021-05-25'
```

add_to_date

```
add_to_date(date, years=0, months=0, weeks=0, days=0, hours=0, minutes=0,
seconds=0, as_string=False, as_datetime=False)
```

```
`date`: A string representation or `datetime` object, uses the current
`datetime` if `None` is passed
`as_string`: Return as string
`as_datetime`: If `as_string` is True and `as_datetime` is also True, returns a
`datetime` string otherwise just the `date` string.
```

This function can be quite handy for doing date/datetime deltas, for instance, adding or substracting certain number of days from a particular date/datetime.

```
from datetime import datetime # from python std library
from frappe.utils import add_to_date

today = datetime.now().strftime('%Y-%m-%d')
print(today) # '2021-05-21'

after_10_days = add_to_date(datetime.now(), days=10, as_string=True)
print(after_10_days) # '2021-05-31'

add_to_date(datetime.now(), months=2) # datetime.datetime(2021, 7, 21, 15, 31,
18, 119999)
add_to_date(datetime.now(), days=10, as_string=True, as_datetime=True) # '2021-
05-31 15:30:23.757661'
add_to_date(None, years=6) # datetime.datetime(2027, 5, 21, 15, 32, 31, 652089)
```

pretty_date

```
pretty_date(iso_datetime)
```

Takes an ISO time and returns a string representing how long ago the date represents. Very common in communication applications like instant messengers.

```
from frappe.utils import pretty_date, now, add_to_date

pretty_date(now()) # 'just now'

# Some example outputs:

# 1 hour ago
# 20 minutes ago
# 1 week ago
# 5 years ago
```

format_duration

`format_duration(seconds, hide_days=False)`

Converts the given duration value in seconds (float) to duration format.

```
from frappe.utils import format_duration

format_duration(50) # '50s'
format_duration(10000) # '2h 46m 40s'
format_duration(1000000) # '11d 13h 46m 40s'

# Convert days to hours
format_duration(1000000, hide_days=True) # '277h 46m 40s'
```

comma_and

`comma_and(some_list, add_quotes=True)`

Given a list or tuple `some_list`, returns a string of the format `1st item, 2nd item, .... and last item`. This function uses `frappe._`, so you don't have to worry about the translations for the word `and`. If `add_quotes` is `False`, returns the items without quotes, with quotes otherwise. If the type of `some_list` passed as an argument is something other than a list or tuple, it (`some_list`) is returned as it is.

```
from frappe.utils import comma_and

comma_and([1, 2, 3]) # "'1', '2' and '3'"
comma_and(['Apple', 'Ball', 'Cat'], add_quotes=False) # 'Apple, Ball and Cat'
comma_and('abcd') # 'abcd'
```

> There is also a `comma_or` function which is similar to `comma_and` except the separator, which is `or` in the case of `comma_or`.

money_in_words

`money_in_words(number, main_currency=None, fraction_currency=None)`

```
`number`: A floating point money amount
`main_currency`: Uses this as the main currency. If not given, tries to fetch
from default settings or uses `INR` if not found there.
```

This function returns string in words with currency and fraction currency.

```
from frappe.utils import money_in_words

money_in_words(900) # 'INR Nine Hundred and Fifty Paisa only.'
money_in_words(900.50) # 'INR Nine Hundred and Fifty Paisa only.'
money_in_words(900.50, 'USD') # 'USD Nine Hundred and Fifty Centavo only.'
```

```
money_in_words(900.50, 'USD', 'Cents') # 'USD Nine Hundred and Fifty Cents
only.'
```

validate_json_string

`validate_json_string(string)`

Raises `frappe.ValidationError` if the given `string` is a valid JSON (JavaScript Object Notation) string. You can use a `try-except` block to handle a call to this function as shown the code snippet below.

```
import frappe
from frappe.utils import validate_json_string

# No Exception thrown
validate_json_string('[]')
validate_json_string('[{}]')
validate_json_string('[{"player": "one", "score": 199}]')

try:
    # Throws frappe.ValidationError
    validate_json_string('invalid json')
except frappe.ValidationError:
    print('Not a valid JSON string')
```

random_string

`random_string(length)`

This function generates a random string containing `length` number of characters. This can be useful for cryptographic or secret generation for some cases.

```
from frappe.utils import random_string

random_string(40) # 'mcrLCrlvkUdkaOe8m5xMI8IwDB8lszwJsWtZFveQ'
random_string(6) # 'htrB4L'
random_string(6) #'HNRirG'
```

unique

```
unique(seq)
```

`seq`: An iterable / Sequence

This function returns a list of elements of the given sequence after removing the duplicates. Also, preserves the order, unlike: `list(set(seq))`.

```python
from frappe.utils import unique

unique([1, 2, 3, 1, 1, 1]) # [1, 2, 3]
unique('abcda') # ['a', 'b', 'c', 'd']
unique(('Apple', 'Apple', 'Banana', 'Apple')) # ['Apple', 'Banana']
```

get_pdf

get_pdf(html, options=None, output=None)

```
`html`: HTML string to render
`options`: An optional `dict` for configuration
`output`: A optional `PdfFileWriter` object.
```

This function uses `pdfkit` and `pyPDF2` modules to generate PDF files from HTML. If `output` is provided, appends the generated pages to this object and returns it, otherwise returns a `byte` stream of the PDF.

For instance, generating and returning a PDF as response to a web request:

```python
import frappe
from frappe.utils.pdf import get_pdf

@frappe.whitelist(allow_guest=True)
def generate_invoice():
    cart = [{
        'Samsung Galaxy S20': 10,
        'iPhone 13': 80
    }]

    html = '<h1>Invoice from Star Electronics e-Store!</h1>'

    # Add items to PDF HTML
    html += '<ol>'
    for item, qty in cart.items():
        html += f'<li>{item} - {qty}</li>'
    html += '</ol>'

    # Attaching PDF to response
    frappe.local.response.filename = 'invoice.pdf'
    frappe.local.response.filecontent = get_pdf(html)
    frappe.local.response.type = 'pdf'
```

get_abbr

get_abbr(string, max_len=2)

Returns an abbrivated (initials only) version of the given `string` with a maximum of `max_len` letters. It is extensively used in Frappe Framework and ERPNext to generate thumbnail or placeholder images.

```
from frappe.utils import get_abbr

get_abbr('Gavin') # 'G'
get_abbr('Coca Cola Company') # 'CC'
get_abbr('Mohammad Hussain Nagaria', max_len=3) # 'MHN'
```

validate_url

`validate_url(txt, throw=False, valid_schemes=None)`

```
`txt`: A string to check validity
`throw`: Weather to throw an exception if `txt` does not represent a valid URL, `False` by default
`valid_schemes`: A string or an iterable (list, tuple or set). If provided, checks the given URL's scheme against this.
```

This utility function can be used to check if a string represents a valid URL address.

```
from frappe.utils import validate_url

validate_url('google') # False
validate_url('https://google.com') # True
validate_url('https://google.com', throw=True) # throws ValidationError
```

validate_email_address

`validate_email_address(email_str, throw=False)`

Returns a string containing the email address or comma-separated list of valid email addresses present in the given `email_str`.
If `throw` is `True`, `frappe.InvalidEmailAddressError` is thrown in case of no valid email address in present in the given string else an empty string is returned.

```
from frappe.utils import validate_email_address

# Single valid email address
validate_email_address('rushabh@erpnext.com') # 'rushabh@erpnext.com'
validate_email_address('other text, rushabh@erpnext.com, some other text') # 'rushabh@erpnext.com'

# Multiple valid email address
validate_email_address(
    'some text, rushabh@erpnext.com, some other text, faris@erpnext.com, yet another no-emailic phrase.'
```

```
) # 'rushabh@erpnext.com, faris@erpnext.com'

# Invalid email address
validate_email_address('some other text') # ''
```

validate_phone_number

`validate_phone_number(phone_number, throw=False)`

Returns `True` if `phone_number` (string) is a valid phone number. If `phone_number` is invalid and `throw` is `True`, `frappe.InvalidPhoneNumberError` is thrown.

```
from frappe.utils import validate_phone_number

# Valid phone numbers
validate_phone_number('753858375') # True
validate_phone_number('+91-75385837') # True

# Invalid phone numbers
validate_phone_number('invalid') # False
validate_phone_number('87345%%', throw=True) # InvalidPhoneNumberError
```

frappe.cache()

`cache()`

Returns the redis connection, which is an instance of class `RedisWrapper` which is inherited from the `redis.Redis` class. You can use this connection to use the Redis cache to store/retrieve key-value pairs.

```
import frappe

cache = frappe.cache()

cache.set('name', 'frappe') # True
cache.get('name') # b'frappe'
```

frappe.sendmail()

`sendmail(recipients=[], sender="", subject="No Subject", message="No Message", as_markdown=False, template=None, args=None, **kwargs)`

```
`recipients`: List of recipients
`sender`: Email sender. Default is current user or default outgoing account
`subject`: Email Subject
`message`: (or `content`) Email Content
`as_markdown`: Convert content markdown to HTML
`template`: Name of html template (jinja) from templates/emails folder
```

```
`args`: Arguments for rendering the template
```

For most cases, the above arguments are sufficient but there are many other keyword arguments that can be passed to this function. To see all the keyword arguments, please have a look the implementation of this function (`frappe/__init__.py`).

This function can be used to send email using user's default **Email Account** or global default **Email Account**.

```
import frappe

recipients = [
    'gavin@erpnext.com',
    'hussain@erpnext.com'
]

frappe.sendmail(
    recipients=recipients,
    subject=frappe._('Birthday Reminder'),
    template='birthday_reminder',
    args=dict(
        reminder_text=reminder_text,
        birthday_persons=birthday_persons,
        message=message,
    ),
    header=_('Birthday Reminder 🎂')
)
```

Sample Jinja template file:

```
<!-- templates/emails/birthday_reminder.html -->
<div>
<div class="gray-container text-center">
<div>
        {% for person in birthday_persons %}
            {% if person.image  %}
                <img src="{{ person.image }}" title="{{ person.name }}"/>
            {% endif %}
        {% endfor %}
    </div>
<div style="margin-top: 15px;">
<span>{{ reminder_text }}</span>
<p class="text-muted">{{ message }}</p>
</div>
</div>
```

Attaching Files

You can easily attach files to your email by passing a list of attachments to the `sendmail` function:

```
frappe.sendmail(
    ["faris@frappe.io", "hussain@frappe.io"],
    message="## hello, *bro*"
    attachments=[{"file_url": "/files/hello.png"}],
    as_markdown=True
)
```

Notice how attachments are a list of dictionaries having a key `file_url`. You can find this `file_url` in a `File` document's `file_url` field.

filelock

File lock can be used to synchron ize processes to avoid race conditions.

Example: Writing to a file can cause race condition if multiple writers are trying to write to a file. So we create a named lock so processes can see the lock and wait until it's avialable for writing.

```
from frappe.utils.synchronization import filelock

def update_important_config(config, file):
    with filelock("config_name"):
        json.dumps(config, file)
```

last updated 2 months ago