Finmars Standard Library

Part of Workflow Engine that exposes to user useful utilities to work with Workflow or Finmars Platform REST API

```
import csv
import datetime
import importlib
import json
import logging
import os
import time
from datetime import timedelta

import jwt
import pandas as pd
import requests
from django.core.files.base import ContentFile
from flatten_json import flatten

from workflow.authentication import FinmarsRefreshToken
from workflow.models import User, Space
from workflow_app import settings

_l = logging.getLogger('workflow')



class DjangoStorageHandler(logging.Handler):
    def __init__(self, log_file, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.log_file = log_file

    def emit(self, record):
        log_entry = self.format(record)

        storage = Storage()
```

```
        storage.append_text(self.log_file, log_entry)

        # with storage.open(self.log_file, 'a') as log_file:
        #     log_file.write(log_entry + '\n')


# DEPRECATED, remove in 1.9.0
def get_access_token(ttl_minutes=60 * 8, *args, **kwargs):
    bot = User.objects.get(username="finmars_bot")

    # Define the expiration time +1 hour from now
    expiration_time = datetime.datetime.utcnow() + datetime.timedelta(minutes=ttl_minutes)

    space = Space.objects.all().first()

    # Define the payload with the expiration time and username
    payload = {
        'username': bot.username,
        'realm_code': space.realm_code,
        'space_code': space.realm_code,
        'exp': expiration_time,
        'iat': datetime.datetime.utcnow()  # Issued at time
    }

    # Encode the JWT token
    jwt_token = jwt.encode(payload, settings.SECRET_KEY, algorithm='HS256')

    token = FinmarsRefreshToken(jwt_token)

    return token


# This one is good
def get_refresh_token(ttl_minutes=60 * 8, *args, **kwargs):
    bot = User.objects.get(username="finmars_bot")

    # Define the expiration time +1 hour from now
    expiration_time = datetime.datetime.utcnow() + datetime.timedelta(minutes=ttl_minutes)

    space = Space.objects.all().first()
```

```
    # Define the payload with the expiration time and username
    payload = {
        'username': bot.username,
        'realm_code': space.realm_code,
        'space_code': space.realm_code,
        'exp': expiration_time,
        'iat': datetime.datetime.utcnow()  # Issued at time
    }

    # Encode the JWT token
    jwt_token = jwt.encode(payload, settings.SECRET_KEY, algorithm='HS256')

    token = FinmarsRefreshToken(jwt_token)

    return token


def get_domain():
    return settings.DOMAIN_NAME


def get_space():
    space = Space.objects.all().first()

    return space


def get_space_code():
    space = Space.objects.all().first()

    return space.space_code


def get_base_path():
    # TODO http or https?
    return 'https://' + get_domain() + '/' + get_realm_code() + '/' + get_space_code()


def get_realm_code():
    space = Space.objects.all().first()
```

```
    return space.realm_code


def create_logger(name, log_format=None):
    if not log_format:
        log_format = "[%(asctime)s][%(levelname)s][%(name)s][%(filename)s:%(funcName)s:%(lineno)d] -
%(message)s"
    formatter = logging.Formatter(log_format)

    log_dir = "/.system/log/"

    log_file = os.path.join(log_dir, str(name) + ".log")
    file_handler = DjangoStorageHandler(log_file)
    file_handler.setFormatter(formatter)

    logger = logging.getLogger(name)
    logger.setLevel(logging.INFO)

    logger.addHandler(file_handler)

    return logger


def execute_expression(expression):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = {
        'expression': expression,
        'is_eval': True
    }

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/utils/expression/'
```

```
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/utils/expression/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def execute_expression_procedure(payload):
    refresh = get_access_token

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/procedures/expression-procedure/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/expression-
procedure/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def execute_data_procedure(payload):
    refresh = get_refresh_token()
```

```
    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/procedures/data-procedure/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/data-
procedure/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def get_data_procedure_instance(id):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/procedures/data-procedure-instance/%s/' % id
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/data-procedure-
instance/%s/' % id
```

```
    response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def execute_pricing_procedure(payload):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/procedures/pricing-procedure/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/pricing-
procedure/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def execute_task(task_name, payload={}):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
```

```
            'Authorization': f'Bearer {refresh.access_token}'}

    data = {
        'task_name': task_name,
        'payload': payload
    }

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/tasks/task/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/tasks/task/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def update_task_status(platform_task_id, status, result=None, error=None):
    refresh = get_refresh_token()
    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
            'Authorization': f'Bearer {refresh.access_token}'}

    data = {
        'status': status,
        'result': result,
        'error': error,
    }

    url = f'{get_base_path()}/api/v1/tasks/task/{platform_task_id}/update-status/'
    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)
    try:
        response.raise_for_status()
        return response.json()
    except Exception as e:
        _l.error("update_task_status error: %s" % e)
```

```
def get_task(id):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/tasks/task/%s/' % id
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/tasks/task/%s/' % id

    response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()



def _wait_task_to_complete_recursive(task_id=None, retries=5, retry_interval=60, counter=None):
    if counter == retries:
        raise Exception("Task exceeded retries %s count" % retries)

    try:
        result = get_task(task_id)

        if result['status'] not in ['progress', 'P', 'I']:
            return result
    except Exception as e:
        _l.error("_wait_task_to_complete_recursive %s" % e)

    counter = counter + 1

    time.sleep(retry_interval)
```

```
    return _wait_task_to_complete_recursive(task_id=task_id, retries=retries, retry_interval=retry_interval,
                                            counter=counter)


def wait_task_to_complete(task_id=None, retries=5, retry_interval=60):
    counter = 0
    result = None

    result = _wait_task_to_complete_recursive(task_id=task_id, retries=retries, retry_interval=retry_interval,
                                              counter=counter)

    return result


def poll_workflow_status(workflow_id, max_retries=100, wait_time=5):
    url = f'/workflow/api/workflow/{workflow_id}/'  # Replace with your actual API endpoint

    for attempt in range(max_retries):
        data = request_api(url)

        if data:
            status = data.get('status')
            _l.info(f'Attempt {attempt + 1}: Workflow status is {status}')

            if status in ['success', 'error']:
                return status  # Return the status when it's success or error

        else:
            _l.error(f'Error fetching status')

        time.sleep(wait_time)  # Wait before the next attempt

    _l.info('Max retries reached. Workflow status not successful.')
    return None  # Indicate that the status was not found


def _wait_procedure_to_complete_recursive(procedure_instance_id=None, retries=5, retry_interval=60,
counter=None):
    if counter == retries:
        raise Exception("Task exceeded retries %s count" % retries)
```

```
    result = get_data_procedure_instance(procedure_instance_id)

    counter = counter + 1

    if result['status'] not in ['progress', 'P', 'I']:
        return result

    time.sleep(retry_interval)

    return _wait_procedure_to_complete_recursive(procedure_instance_id=procedure_instance_id, retries=retries,
                                                 retry_interval=retry_interval, counter=counter)


def wait_procedure_to_complete(procedure_instance_id=None, retries=5, retry_interval=60):
    counter = 0
    result = None

    result = _wait_procedure_to_complete_recursive(procedure_instance_id=procedure_instance_id,
retries=retries,
                                                   retry_interval=retry_interval, counter=counter)

    return result


def execute_transaction_import(payload):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}
    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/import/transaction-import/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/import/transaction-
```

```
import/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def execute_simple_import(payload):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/import/simple-import/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/import/simple-import/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def request_api(path, method='get', data=None):
    refresh = get_refresh_token()

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}
```

```
    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code + path
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + path

    response = None

    if method.lower() == 'get':

        response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

    elif method.lower() == 'post':

        response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    elif method.lower() == 'put':

        response = requests.put(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    elif method.lower() == 'patch':

        response = requests.patch(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    elif method.lower() == 'delete':

        response = requests.delete(url=url, headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200 and response.status_code != 201 and response.status_code != 204:
        raise Exception(response.text)

    if response.status_code != 204:
        return response.json()

    return {"status": "no_content"}


class Storage():
```

```
def __init__(self):

    from workflow.storage import get_storage

    self.storage = get_storage()

def get_base_path(self):

    space = Space.objects.all().first()

    return space.space_code

def listdir(self, path):
    return self.storage.listdir('/' + self.get_base_path() + path)

def open(self, name, mode='rb'):

    # TODO permission check

    if name[0] == '/':
        name = self.get_base_path() + name
    else:
        name = self.get_base_path() + '/' + name

    return self.storage.open(name, mode)

def read_json(self, filepath, mode='rb'):
    with self.open(filepath, mode) as state:
        state_content = json.loads(state.read())
    return state_content

def read_csv(self, filepath, mode='rb'):
    with self.open(filepath, mode) as f:
        reader = csv.DictReader(f)
        data = list(reader)
    return data

def read(self, filepath, mode='rb'):

    # Open the file from your storage backend
    file_obj = self.open(filepath, mode)  # 'rb' is to read in binary mode
```

```
    try:
        # Read the file's contents
        file_content = file_obj.read()
        return file_content
    finally:
        # Make sure we close the file object
        file_obj.close()

def delete(self, name):

    # TODO permission check

    if name[0] == '/':
        name = self.get_base_path() + name
    else:
        name = self.get_base_path() + '/' + name

    return self.storage.delete(name)

def exists(self, name):

    # TODO permission check

    if name[0] == '/':
        name = self.get_base_path() + name
    else:
        name = self.get_base_path() + '/' + name

    return self.storage.exists(name)

def save(self, name, content):

    if name[0] == '/':
        name = self.get_base_path() + name
    else:
        name = self.get_base_path() + '/' + name

    return self.storage.save(name, content)

def save_text(self, name, content):
```

```
        if name[0] == '/':
            name = self.get_base_path() + name
        else:
            name = self.get_base_path() + '/' + name

        return self.storage.save(name, ContentFile(content.encode('utf-8')))

    def append_text(self, name, content):

        if self.storage.exists(name):
            with self.open(name, 'r') as file:
                content = file.read()
                content = content + content + '\n'

        return self.storage.save(name, ContentFile(content.encode('utf-8')))


class Utils():

    def get_current_space_code(self):
        space = Space.objects.all().first()
        return space.space_code

    def get_list_of_dates_between_two_dates(self, date_from, date_to, to_string=False):
        result = []
        format = '%Y-%m-%d'

        if not isinstance(date_from, datetime.date):
            date_from = datetime.datetime.strptime(date_from, format).date()

        if not isinstance(date_to, datetime.date):
            date_to = datetime.datetime.strptime(date_to, format).date()

        diff = date_to - date_from

        for i in range(diff.days + 1):
            day = date_from + timedelta(days=i)
            if to_string:
                result.append(str(day))
            else:
```

```
            result.append(day)

    return result

def is_business_day(self, date):
    return bool(len(pd.bdate_range(date, date)))

def get_yesterday(self, ):
    today = datetime.now()
    yesterday = today - timedelta(days=1)
    return yesterday

def get_list_of_business_days_between_two_dates(self, date_from, date_to, to_string=False):
    result = []
    format = '%Y-%m-%d'

    if not isinstance(date_from, datetime.date):
        date_from = datetime.datetime.strptime(date_from, format).date()

    if not isinstance(date_to, datetime.date):
        date_to = datetime.datetime.strptime(date_to, format).date()

    diff = date_to - date_from

    for i in range(diff.days + 1):
        day = date_from + timedelta(days=i)

        if self.is_business_day(day):

            if to_string:
                result.append(str(day))
            else:
                result.append(day)

    return result

def import_from_storage(self, file_path):
    # get the directory and the filename without extension

    space = get_space()
```

```
        if file_path[0] == '/':
            file_path = os.path.join(settings.WORKFLOW_STORAGE_ROOT + '/tasks/' + space.space_code +
file_path)
        else:
            file_path = os.path.join(settings.WORKFLOW_STORAGE_ROOT + '/tasks/' + space.space_code + '/' +
file_path)

        _l.info('import_from_storage.file_path %s' % file_path)

        directory, filename = os.path.split(file_path)
        module_name, _ = os.path.splitext(filename)

        _l.info('import_from_storage.module_name %s' % module_name)
        _l.info('import_from_storage.file_path %s' % file_path)

        loader = importlib.machinery.SourceFileLoader(module_name, file_path)
        module = loader.load_module()

        # add the directory to sys.path
        # spec = importlib.util.spec_from_file_location(module_name, file_path)
        #
        # if spec is None:
        #     raise ImportError(f"Cannot import file {filename}")
        #
        # module = importlib.util.module_from_spec(spec)
        #
        # # execute the module
        # spec.loader.exec_module(module)
        #
        # # return the module
        return module

    def relative_import_from_storage(self, file_path, base_path):

        """
        Imports a module from a given file path, resolving the path from a specified base path.

        :param file_path: Relative or absolute path to the Python file to import.
        :param base_path: Base directory against which relative paths should be resolved.
        :return: The imported module.
        """
```

```
    # Resolve the relative file_path against the provided base directory
    absolute_file_path = os.path.normpath(os.path.join(base_path, file_path))

    # _l.info(f'Normalized file path: {absolute_file_path}')

    # Continue with your existing logic, but use absolute_file_path instead of file_path
    directory, filename = os.path.split(absolute_file_path)
    module_name, _ = os.path.splitext(filename)

    # _l.info(f'import_from_storage.module_name {module_name}')
    # _l.info(f'import_from_storage.file_path {absolute_file_path}')

    loader = importlib.machinery.SourceFileLoader(module_name, absolute_file_path)
    module = loader.load_module()

    # add the directory to sys.path
    # spec = importlib.util.spec_from_file_location(module_name, file_path)
    #
    # if spec is None:
    #     raise ImportError(f"Cannot import file {filename}")
    #
    # module = importlib.util.module_from_spec(spec)
    #
    # # execute the module
    # spec.loader.exec_module(module)
    #
    # # return the module
    return module

def tree_to_flat(self, data, **kwargs):

    return flatten(data, **kwargs)

# Example conversions:
# "Héllo World!"        -> "hello_world!"
# "Café.com"            -> "cafe_com"
# "Jürgen.Smith"        -> "jurgen_smith"
# "Mañana es jueves."   -> "manana_es_jueves_"
# "Gérard Dépardieu"    -> "gerard_depardieu"
# "naïve artist"        -> "naive_artist"
```

```
    # Problem here Example conversions with different accents on 'e':
    # "é" -> "e"
    # "è" -> "e"
    # "ê" -> "e"
    # "ë" -> "e"
    # Example conversions:
    # "école"        -> "U233cole"
    # "café.com"     -> "cafeU233_com"
    # "Jürgen.Smith" -> "jU252rgen_smith"
    # "élève"        -> "U233lU232ve"
    # "Mañana"       -> "manU241ana"
    # "Gödel"        -> "gU246del"
    def convert_to_ascii(self, input_string):
        # Convert the input string to lowercase
        input_string = input_string.lower()

        # Convert spaces and dots to underscores
        modified_string = input_string.replace(' ', '_').replace('.', '_')

        # Function to convert each character
        def to_ascii_or_unicode(char):
            try:
                # Try to encode the character in ASCII
                ascii_char = char.encode('ascii')
                return ascii_char.decode()  # Return as string if it's a valid ASCII character
            except UnicodeEncodeError:
                # If it's not an ASCII character, return its Unicode code point
                return f"U{ord(char)}"

        # Apply the conversion to each character in the string
        ascii_string = ''.join(to_ascii_or_unicode(c) for c in modified_string)

        return ascii_string


class Vault():

    # hashicorp
    # finmars
    def get_secret(self, path, provider="finmars"):
        refresh = get_refresh_token()  # TODO refactor, should be permission check
```

```
        # _l.info('refresh %s' % refresh.access_token)

        if provider == 'finmars':

            # pieces = path.split('/')
            # engine_name = pieces[0]
            # secret_path = pieces[1]

            headers = {'Content-type': 'application/json', 'Accept': 'application/json',
                       'Authorization': f'Bearer {refresh.access_token}'}

            space = get_space()

            url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
f'/api/v1/vault/vault-record/?user_code=' + path

            response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

            if response.status_code != 200:
                raise Exception(response.text)

            data = response.json()

            secret_data = None

            for item in data['results']:

                if path == item['user_code']:
                    secret_data = item['data']

            if not secret_data:
                raise Exception(f"Secret is {path} not found")

            return secret_data

        elif provider == 'hashicorp':

            pieces = path.split('/')
            engine_name = pieces[0]
            secret_path = pieces[1]
```

```
            headers = {'Content-type': 'application/json', 'Accept': 'application/json',
                       'Authorization': f'Bearer {refresh.access_token}'}

            space = get_space()

            if space.realm_code and space.realm_code != 'realm00000':
                url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
f'/api/v1/vault/vault-secret/get/?engine_name={engine_name}&path={secret_path}'
            else:
                url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + f'/api/v1/vault/vault-
secret/get/?engine_name={engine_name}&path={secret_path}'

            response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

            if response.status_code != 200:
                raise Exception(response.text)

            return response.json()['data']['data']

        else:
            raise Exception("Unknown provider %s" % provider)


storage = Storage()

utils = Utils()

vault = Vault()
```

Revision #1
Created 3 November 2024 12:08:48 by Sergei Zhitenev
Updated 3 November 2024 12:09:44 by Sergei Zhitenev