Capitains.Nautilus Documentation

Release 0.0.1

Thibault Clérice

February 15, 2017

Contents

CHAPTER 1

Capitains Nautilus

Documentation

Documentation will be built in time.

Running Nautilus from the command line

This small tutorial takes that you have one or more Capitains formated repositories (such as http://github.com/PerseusDL/canonical-latinLit) in the folders /home/USERNAME/repository1 where USERNAME is your user session name.

1. (Advised) Create a virtual environment and source it : `virtualenv -p /usr/bin/python3 env`, `source env/bin/activate`
2. **With development version:**
    - Clone the repository : `git clone https://github.com/Capitains/Nautilus.git`
    - Go to the directory : `cd Nautilus`
    - Install the source with develop option : `python setup.py develop`
2. **With production version (not available for now):**
    - Install from pip : `pip install capitains-nautilus`
3. You will be able now to call capitains nautilus help information through `capitains-nautilus --help`
4. Basic setting for testing a directory is `capitains-nautilus --debug /home/USERNAME/repository1`. This can take more than one repository at the end such as `capitains-nautilus --debug /home/USERNAME/repository1 /home/USERNAME/repository2`. You can force host and port through –host and –port parameters.

Source for the tests

Textual resources and inventories are owned by Perseus under CC-BY Licences. See https://github.com/PerseusDL/canonical-latinLit and https://github.com/PerseusDL/canonical-farsiLit

Contents

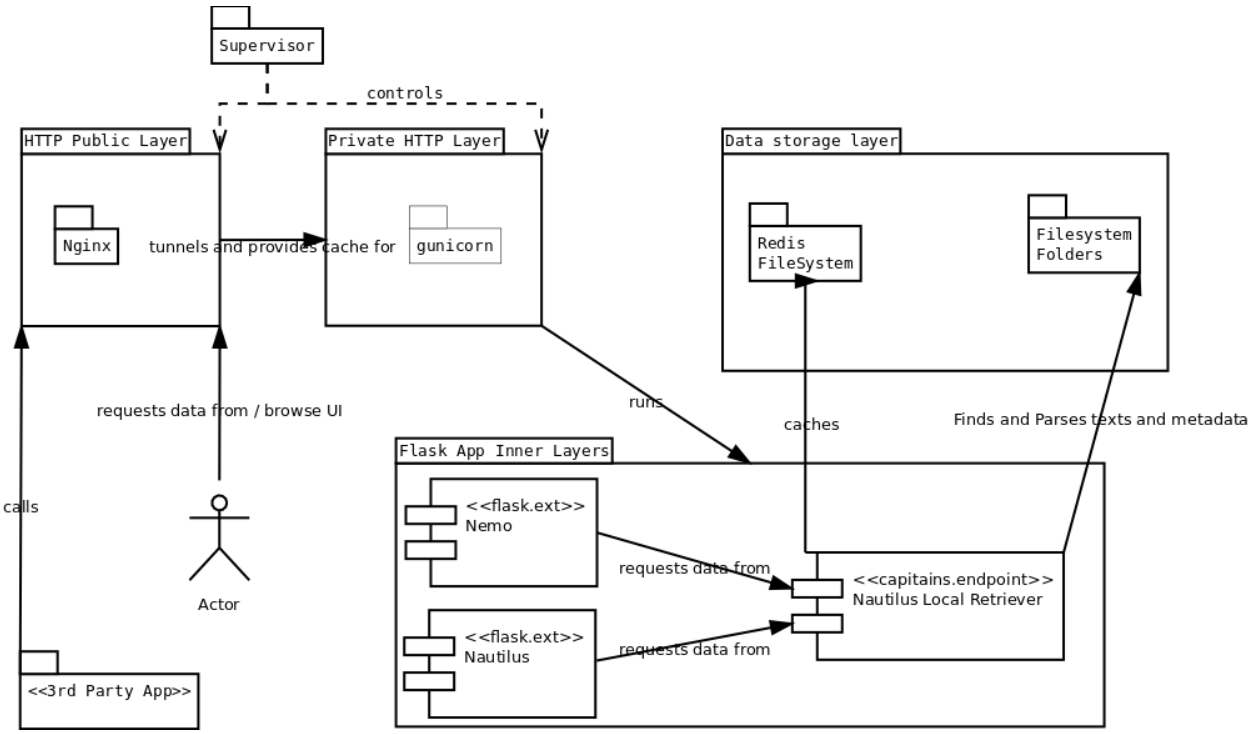
Nautilus' Production Environment deployment advices

Environment

We recommend highly to use Debian based configuration as they are the only one having been tested for now. The following configuration takes into account what we think might be the best configuration available with a good cache system.

You can use a docker image we built and fork it for your own use. As of *April 11th, 2016*, the **docker image does not use Redis-based cache** but filesystem based cache.

The environment we propose contains a flask.ext.nemo instance, for control purposes. Disabling it is documented.



Nginx, Supervisor, GUnicorn

The following configuration file for supervisor is enough for running the whole nginx, supervisord and gunicorn trio. In general, servers configuration should always - in production - have a trio made of an HTTP/Reverse proxy server such as *nginx*, a process control system such as *supervisor* and a WSGI http server such as *gunicorn* or *uWSGI*. This configuration takes the road of *gunicorn*, but feel free to test and benchmark any combination to know what works best on your own server(s).

On the software is read, run :

```
service supervisor stop
service nginx stop
```

Setup

```
apt-get install zlib1g-dev libxslt1-dev libxml2-dev python3 python3-dev python3-pip build-essential m
apt-get install python-setuptools

easy_install pip
pip2.7 install supervisor-stdout
easy_install3 --upgrade pip

mkdir /var/capitains-server
cd /var/capitains-server

virtualenv venv
pip install Nautilus
pip install gunicorn
pip install flask_nemo
```

You'll need then to create your own app (You can see below for an example)

Configurations files

Warning: These configuration files are designed for the specified directories and services

Listing 1.1: /etc/supervisord.conf

```
[supervisord]
nodaemon = true

[program:nginx]
command = /usr/sbin/nginx
startsecs = 5
stdout_events_enabled = true
stderr_events_enabled = true

[program:app-gunicorn]
# See explanation for this line
command=/usr/local/bin/gunicorn app:app -w 4 --threads 2 -b 127.0.0.1:5000 --log-level=debug --python
directory=/code
stdout_events_enabled = true
stderr_events_enabled = true

[eventlistener:stdout]
command = supervisor_stdout
buffer_size = 100
events = PROCESS_LOG
result_handler = supervisor_stdout:event_handler
```

Listing 1.2: /etc/nginx/nginx.conf

```
daemon off;
error_log /dev/stdout info;
worker_processes 1;

# user nobody nogroup;
```

```
pid /tmp/nginx.pid;

events {
    worker_connections 1024;
    accept_mutex off;
}

http {
    include mime.types;
    default_type application/octet-stream;
    access_log /dev/stdout combined;
    sendfile on;

    upstream app_server {
        # For a TCP configuration:
        server 127.0.0.1:5000 fail_timeout=0;
    }

    server {
        listen 80 default;
        client_max_body_size 4G;
        server_name _;

        keepalive_timeout 5;

        # path for static files
        root /opt/app/static;

        location / {
            # checks for static file, if not found proxy to app
            try_files $uri @proxy_to_app;
        }

        location @proxy_to_app {
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $http_host;
            proxy_redirect off;

            proxy_pass   http://app_server;
        }

    }
}
```

Flask Application Configuration

Nemo And FileSystemCache (Easy to maintain)

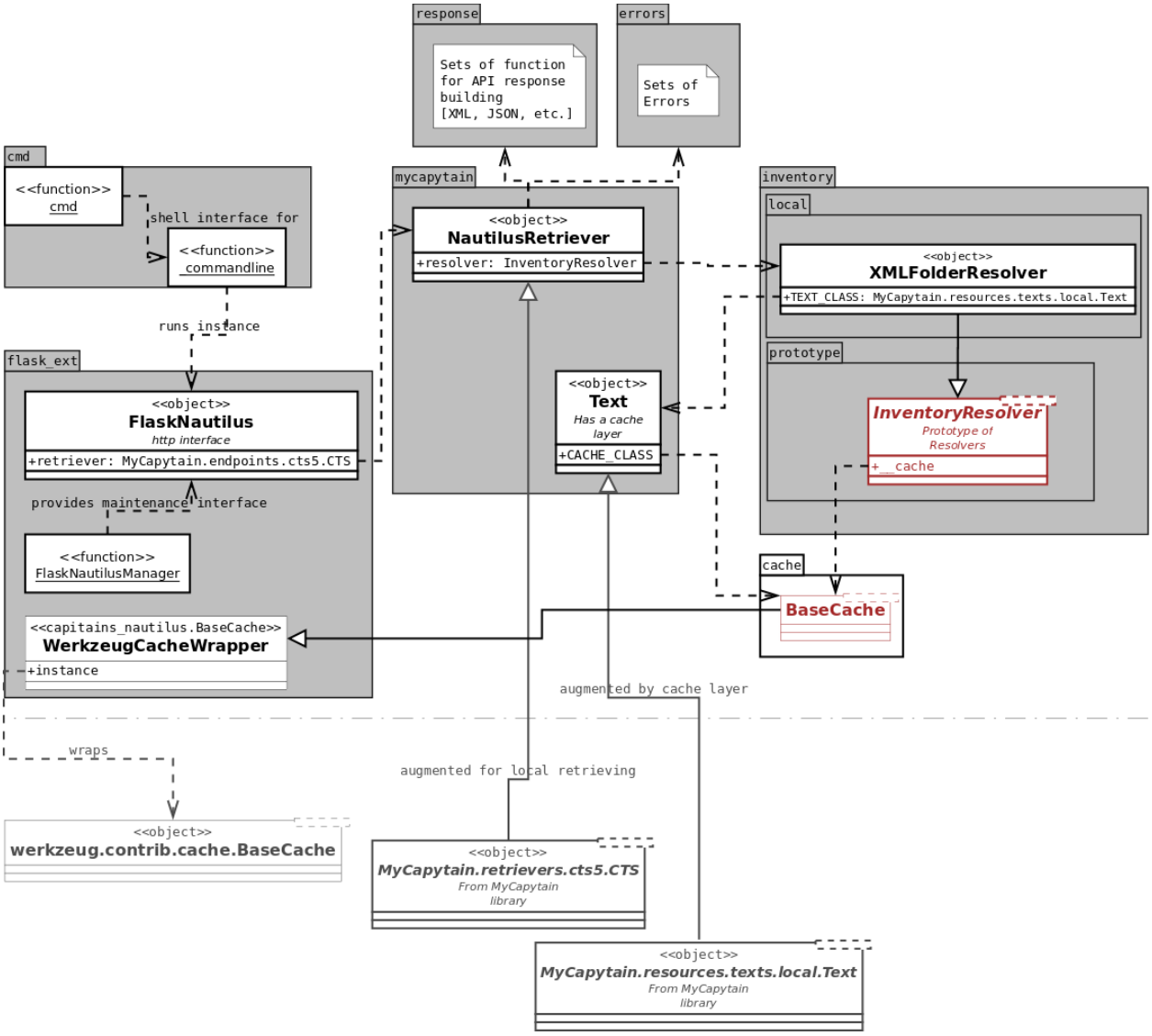
The following configuration is based on a FileSystemCache. This means that you do not need to install, run and maintain more advanced Cache system such as Redis. This also means this should be slower. The implementation contains a frontend, you should be able to run it without it.

```
# -*- coding: utf-8 -*-

from flask import Flask, request
from flask.ext.nemo import Nemo
```

```
from capitains_nautilus.flask_ext import FlaskNautilus
from werkzeug.contrib.cache import FileSystemCache
from flask_cache import Cache

app = Flask("Nautilus")
nautilus_cache = FileSystemCache("/var/capitains-cache")
nautilus = FlaskNautilus(
    app=app,
    prefix="/api/cts",
    name="nautilus",
    # Add here paths to all CapiTainS repository you have locally
    resources=["/var/capitains-data/canonical-latinLit-master"],
    parser_cache=nautilus_cache,
    http_cache=Cache(config={'CACHE_TYPE': "simple"})
)
# We set up Nemo
# This part can be removed
nemo = Nemo(
    app=app,
    name="nemo",
    base_url="",
    api_url="/api/cts",
    endpoint=nautilus.retriever
)
# We register its routes
nemo.register_routes()
# We register its filters
nemo.register_filters()

# Removes this line for production
app.debug = True

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0')
```

Capitains Nautilus API Documentation

Library Structure



Resolvers

Resolver provides a system to retrieve a text file and an inventory from local resources for example.

CapiTainS formatted repository

class `capitains_nautilus.inventory.local.`**`XMLFolderResolver`**(*resource*, *inventories=None*, *cache=None*, *name=None*, *logger=None*, *auto_parse=True*)

XML Folder Based resolver.

Parameters

- **resource** (*[str]*) – Resource should be a list of folders retaining data as Capitains Guidelines Repositories
- **name** (*str*) – Key used to differentiate Repository and thus enabling different repo to be used
- **inventories** –
- **cache** (BaseCache) – Cache object to be used for the inventory
- **logger** (*logging*) – Logging object

Variables

- ***TEXT_CLASS*** – Text Class [not instantiated] to be used to parse Texts. Can be changed to support Cache for example
- **inventory_cache_key** – Werkzeug Cache key to get or set cache for the TextInventory
- **texts_cache_key** – Werkzeug Cache key to get or set cache for lists of metadata texts objects
- **texts_parsed** – Werkzeug Cache key to get or set cache for lists of parsed texts objects
- **texts** – List of Text Metadata objects
- **source** – Original resource parameter

Warning: This resolver does not support inventories

TEXT_CLASS
alias of Text

cache(*inventory*, *texts*)
Cache main objects of the resolver : TextInventory and Texts Metadata objects

Parameters

- **inventory** (*TextInventory*) – Inventory resource
- **texts** (*[MyCapytain.resources.inventory.Text]*) – List of Text Metadata Objects

cache_to_text(*urn*)
Get a text from Cache

Parameters **text** – Text to be cached

Returns Text object

Return type *Text*

flush()
Flush current resolver objects and cache

getCapabilities(*urn=None*, *page=None*, *limit=None*, *inventory=None*, *lang=None*, *category=None*, *pagination=True*)
Retrieve a slice of the inventory filtered by given arguments

Parameters

- **urn** (*str*) – Partial URN to use to filter out resources
- **page** (*int*) – Page to show
- **limit** (*int*) – Item Per Page

- **inventory** (*str*) – Inventory name
- **lang** (*str*) – Language to filter on
- **category** (*str*) – Type of elements to show
- **pagination** (*bool*) – Activate pagination

Returns ([Matches], Page, Count)

Return type ([Text], int, int)

getText(*urn*)
Returns a Text object

Parameters **urn** (*str, URN*) – URN of a text to retrieve

Returns Textual resource and metadata

Return type (text.Text, inventory.Text)

parse(*resource*, *cache=True*)
Parse a list of directories ans

Parameters

- **resource** – List of folders
- **cache** – Auto cache the results

Returns An inventory resource and a list of Text metadata-objects

text_to_cache(*text*)
Cache a text

Parameters **text** – Text to be cached

xmlparse(*file*)
Parse a XML file

Parameters **file** – Opened File

Returns Tree

Prototype

class capitains_nautilus.inventory.proto.**InventoryResolver**(*resource*, *auto_parse=True*)
Inventory Resolver Prototype. It is used to serve local xml files and an inventory.

Parameters

- **resource** – Resource used to retrieve texts
- **auto_parse** – Automatically parse the resource on initialization

Variables

- **DEFAULT_PAGE** – Default Page to show
- **PER_PAGE** – Tuple representing the minimal number of texts returned, the default number and the maximum number of texts returned
- **source** – Reading access to original resource
- **texts** – List of MyCapytain.resources.inventory.Text

static **pagination**(*page*, *limit*, *length*)
Help for pagination

Parameters

- **page** – Provided Page
- **limit** – Number of item to show
- **length** – Length of the list to paginate

Returns (Start Index, End Index, Page Number, Item Count)

Retriever

Extension of MyCapytains resources

class capitains_nautilus.mycapytain.**NautilusRetriever**(*folders=None*, *cache=None*, *pagination=True*, *logger=None*, *auto_parse=True*, *resolver=<class 'capitains_nautilus.inventory.local.XMLFolderResolver'>*)

Bases: MyCapytain.retrievers.cts5.CTS

Nautilus Implementation of MyCapytain Endpoint

Parameters

- **folders** (*list (str)*) – List of Capitains Guidelines structured folders
- **logger** (*logging*) – Logging handler
- **auto_parse** – Parses on first execution the resources given to build inventory
- **resolver** (XMLFolderResolver) – Resolver to be used

Variables

- **logger** – Logging handler
- **resolver** – Resolver for repository and text path

getCapabilities(*inventory=None*, *output='text/xml'*, ***kwargs*)
Retrieve the inventory information of an API

Parameters

- **inventory** (*text*) – Name of the inventory
- **format** (*str*) – Format type of response. *capitains_nautilus.response*

Returns Formatted output of the inventory

Return type str

getFirstUrn(*urn*, *inventory=None*, *output='text/xml'*)
Retrieve valid first URN

Parameters

- **urn** (*text*) – URN identifying the text
- **inventory** (*text*) – Name of the inventory

Returns Formatted response with first URN

Return type str

getLabel(*urn*, *inventory=None*, *output='text/xml'*)
Retrieve label informations

Parameters

- **urn** (*text*) – URN identifying the text
- **inventory** (*text*) – Name of the inventory

Returns Formatted response with metadata

Return type str

getPassage(*urn*, *inventory=None*, *context=None*, *output='text/xml'*)
Get a Passage from the repository

Parameters

- **urn** – URN identifying the passage
- **inventory** (*text*) – Name of the inventory
- **format** (*str*) – Format type of response. *capitains_nautilus.response*
- **context** – Unused parameter for now

Returns Passage asked for, in given format

getPassagePlus(*urn*, *inventory=None*, *context=None*, *output='text/xml'*)
Get a Passage and its metadata from the repository

Parameters

- **urn** – URN identifying the passage
- **inventory** (*text*) – Name of the inventory
- **format** (*str*) – Format type of response. *capitains_nautilus.response*
- **context** – Unused parameter for now

Returns Passage asked for, in given format

getPrevNextUrn(*urn*, *inventory=None*, *output='text/xml'*)
Retrieve valid previous and next URN

Parameters

- **urn** (*text*) – URN identifying the text
- **inventory** (*text*) – Name of the inventory

Returns Formatted response with prev and next urn

Return type str

getText(*urn*, *inventory=None*)
Retrieves a text in the inventory in case of partial URN or throw error when text is not accessible

Parameters

- **urn** (*text*) – URN identifying the text
- **inventory** – Name of

Returns (Original URN, Corrected URN, Text, Metadata Text)

getValidReff(*urn*, *inventory=None*, *level=1*, *output='text/xml'*)
Retrieve valid urn-references for a text

Parameters

- **urn** (*text*) – URN identifying the text
- **inventory** (*text*) – Name of the inventory
- **level** (*int*) – Depth of references expected

Returns Formatted response or list of references

Return type str

class capitains_nautilus.mycapytain.**Text**(**args*, ***kwargs*)
Bases: MyCapytain.resources.texts.local.Text

CACHE_CLASS
alias of BaseCache

TIMEOUT = {'getValidReff': 604800}

getValidReff(*level=1*, *reference=None*)
Cached method of the original object

Parameters

- **level** –
- **reference** – Reference object

Returns References

Responses builders

Response generator for the queries

capitains_nautilus.response.**getcapabilities**(*texts*, *page=None*, *count=None*, *output='text/xml'*, ***kwargs*)
Transform a list of texts into a string representation

Parameters **texts** – List of Text objects

Returns String representation of the Inventory

capitains_nautilus.response.**getfirst**(*passage*, *request_urn*, *output='text/xml'*)

capitains_nautilus.response.**getlabel**(*metadata*, *full_urn*, *request_urn*, *output='text/xml'*)

capitains_nautilus.response.**getpassage**(*passage*, *metadata*, *request_urn*, *output='text/xml'*)

capitains_nautilus.response.**getpassageplus**(*passage*, *metadata*, *request_urn*, *output='text/xml'*)

capitains_nautilus.response.**getprevnext**(*passage*, *request_urn*, *output='text/xml'*)

capitains_nautilus.response.**getvalidreff**(*reffs*, *level*, *request_urn*, *output='text/xml'*)

Errors

exception capitains_nautilus.errors.**CTSError**
Bases: BaseException

CODE = None

exception capitains_nautilus.errors.**InvalidContext**
Bases: *capitains_nautilus.errors.CTSError*

Invalid value for context parameter in GetPassage or GetPassagePlus request

CODE = 5

exception capitains_nautilus.errors.**InvalidLevel**
Bases: *capitains_nautilus.errors.CTSError*

Invalid value for level parameter in GetValidReff request

CODE = 4

exception capitains_nautilus.errors.**InvalidURN**
Bases: *capitains_nautilus.errors.CTSError*

Syntactically valid URN refers in invalid value

CODE = 3

exception capitains_nautilus.errors.**InvalidURNSyntax**
Bases: *capitains_nautilus.errors.CTSError*

Invalid URN syntax

CODE = 2

exception capitains_nautilus.errors.**MissingParameter**
Bases: *capitains_nautilus.errors.CTSError*

Request missing one or more required parameters

CODE = 1

exception capitains_nautilus.errors.**UnknownResource**
Bases: *capitains_nautilus.errors.CTSError*

Resource requested is not found

CODE = 6

Cache

class capitains_nautilus.cache.**BaseCache**(*default_timeout=300*)
Bases: object

Based on the werkzeug.contrib.cache.BaseCache object. Provides a wrapper for other cache system in the future.

Parameters **default_timeout** – the default timeout (in seconds) that is used if no timeout is specified on *set()*. A timeout of 0 indicates that the cache never expires.

add(*key*, *value*, *timeout=None*)
Works like *set()* but does not overwrite the values of already existing keys. :param key: the key to set :param value: the value for the key :param timeout: the cache timeout for the key in seconds (if not

specified, it uses the default timeout). A timeout of 0 idicates that the cache never expires.

Returns Same as *set()*, but also `False` for already existing keys.

Return type boolean

clear()
Clears the cache. Keep in mind that not all caches support completely clearing the cache. :returns: Whether the cache has been cleared. :rtype: boolean

dec(*key*, *delta=1*)
Decrements the value of a key by *delta*. If the key does not yet exist it is initialized with *-delta*. For supporting caches this is an atomic operation. :param key: the key to increment. :param delta: the delta to subtract. :returns: The new value or *None* for backend errors.

delete(*key*)
Delete *key* from the cache. :param key: the key to delete. :returns: Whether the key existed and has been deleted. :rtype: boolean

delete_many(**keys*)
Deletes multiple keys at once. :param keys: The function accepts multiple keys as positional

arguments.

Returns Whether all given keys have been deleted.

Return type boolean

get(*key*)
Look up key in the cache and return the value for it. :param key: the key to be looked up. :returns: The value if it exists and is readable, else `None`.

get_dict(**keys*)

Like *get_many()* but return a dict:: d = cache.get_dict("foo", "bar") foo = d["foo"] bar = d["bar"]

Parameters **keys** – The function accepts multiple keys as positional arguments.

get_many(**keys*)
Returns a list of values for the given keys. For each key a item in the list is created:

```
foo, bar = cache.get_many("foo", "bar")
```

Has the same error handling as *get()*. :param keys: The function accepts multiple keys as positional

arguments.

has(*key*)
Checks if a key exists in the cache without returning it. This is a cheap operation that bypasses loading the actual data on the backend. This method is optional and may not be implemented on all caches. :param key: the key to check

inc(*key*, *delta=1*)
Increments the value of a key by *delta*. If the key does not yet exist it is initialized with *delta*. For supporting caches this is an atomic operation. :param key: the key to increment. :param delta: the delta to add. :returns: The new value or `None` for backend errors.

set(*key*, *value*, *timeout=None*)
Add a new key/value to the cache (overwrites value, if key already exists in the cache). :param key: the key to set :param value: the value for the key :param timeout: the cache timeout for the key in seconds (if not

specified, it uses the default timeout). A timeout of 0 idicates that the cache never expires.

Returns `True` if key has been updated, `False` for backend errors. Pickling errors, however, will raise a subclass of `pickle.PickleError`.

Return type boolean

set_many(*mapping*, *timeout=None*)
Sets multiple keys and values from a mapping. :param mapping: a mapping with the keys/values to set. :param timeout: the cache timeout for the key in seconds (if not

specified, it uses the default timeout). A timeout of 0 idicates that the cache never expires.

Returns Whether all given keys have been set.

Return type boolean

Flask Extension

class capitains_nautilus.flask_ext.**FlaskNautilus**(*prefix=''*, *app=None*, *name=None*, *resources=None*, *parser_cache=None*, *compresser=True*, *http_cache=None*, *pagination=False*, *access_Control_Allow_Origin=None*, *access_Control_Allow_Methods=None*, *logger=None*, *auto_parse=True*)

Bases: object

Initiate the class

Parameters

- **prefix** – Prefix on which to install the extension
- **app** – Application on which to register
- **name** – Name to use for the blueprint
- **resources** (*list (str)*) – List of directory to feed the inventory
- **logger** (*logging*) – Logging handler.
- **parser_cache** (BaseCache) – Cache object
- **http_cache** – HTTP Cache should be a FlaskCache object
- **auto_parse** – Parses on first execution the resources given to build inventory. Not recommended for production

Variables

- ***ROUTES*** – List of triple length tuples
- ***Access_Control_Allow_Methods*** – Dictionary with route name and allowed methods over CORS
- ***Access_Control_Allow_Origin*** – Dictionary with route name and allowed host over CORS or "*"
- ***LoggingHandler*** – Logging handler to be set for the blueprint
- **logger** – Logging handler
- **retriever** – CapiTainS Retriever

Access_Control_Allow_Methods = {'r_dispatcher': 'OPTIONS, GET'}

Access_Control_Allow_Origin = '*'

LoggingHandler
alias of StreamHandler

ROUTES = [('/', 'r_dispatcher', ['GET'])]

init_app(*app*, *compresser=False*)
Initiate the extension on the application

Parameters **app** – Flask Application

Returns Blueprint for Flask Nautilus registered in app

Return type Blueprint

init_blueprint()
Properly generates the blueprint, registering routes and filters and connecting the app and the blueprint

Returns Blueprint of the extension

Return type Blueprint

r_dispatcher()
Actual main route of CTS APIs. Transfer typical requests through the ?request=REQUESTNAME route

Returns Response

setLogger(*logger*)
Set up the Logger for the application

Parameters **logger** – logging.Logger object

Returns Logger instance

view(*function_name*)
Builds response according to a function name

Parameters **function_name** – Route name / function name

Returns Function

capitains_nautilus.flask_ext.**FlaskNautilusManager**(*nautilus*, *app=None*)
Provides a manager for flask scripts to perform specific maintenance operations

Parameters

- **nautilus** – Nautilus Extension Instance
- **app** – Flask Application

Returns Sub-Manager

Return type Manager

Import with

class capitains_nautilus.flask_ext.**WerkzeugCacheWrapper**(*instance=None*, **args*, ***kwargs*)
Bases: *capitains_nautilus.cache.BaseCache*

Werkzeug Cache Wrapper for Nautilus Base Cache object

Parameters **instance** – Werkzeug Cache instance

add(*key*, *value*, *timeout=None*)

clear()

delete(*key*)

get(*key*)

set(*key*, *value*, *timeout=None*)

Commandline

capitains_nautilus.cmd.**cmd**()

Indices and tables

- Importing Modules
- genindex
- modindex
- search

C

H

I

L

M

N

P

R

S

T

U

V

W

X