Flask.ext.Nemo Documentation

Release 0.0.1

Thibault Clérice

May 11, 2016

Contents

Capitains Nemo is an User Interface built around the need to make CTS a easy to use, human readable standard for texts. Capitains Nemo counts multiple language implementation, including this one in Python for Flask. Presentend as a classic Flask Extension, *flask.ext.nemo* intends to be a simple, customizable interface between your enduser and your CTS5 API.

The Flask's extension Nemo can be customized from its stylesheets to its functionalities. Adding routes or removing them is as easy as adding a XSL Stylesheet to transform the very own result of a CTS GetPassage results to your own expected output.

CHAPTER 1

Install

You can now install it with pip : *pip install flask_nemo*

If you want to install the latest version, please do the following

```
git clone https://github.com/Capitains/flask-capitains-nemo.git
cd flask-capitains-nemo
virtualenv -p /path/to/python3 venv
source venv/bin/activate
python setup.py install
```

If you have trouble with dependency conflicts with MyCapitains, try running `pip install MyCapytain` this before install

CHAPTER 2

Running Nemo from the command line

This small tutorial takes that you have a CTS API endpoint available, here `http://localhost:8000`

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/flask-capitains-nemo.git`
    - Go to the directory : `cd Nemo`
    - Install the source with develop option : `python setup.py develop`
2. **With production version:**
    - Install from pip : `pip install flask_nemo`
3. You will be able now to call capitains nemo help information through `capitains-nemo --help`
4. Basic setting for testing an api is `capitains-nemo http://localhost:8000`.

2.1 Examples

2.1.1 Simple Configuration

User story [1]

A **researcher** , an **engineer** or both is interested in **CTS** but has **no time** to develop their own application and their own theme : *flask.ext.nemo* will provide a simple, easy to use interface that you can deploy on any server. Even with a really limited knowledge of python.

User story [2]

A **researcher**, an **engineer** or both has already a CTS endpoint and wants to check the output and the browsing system visually.

The simplest configuration of Nemo, or close to it, is to simply give an endpoint url to your Nemo extension, the app you are using and the name of a CTS inventory (if required). This will run a browsing interface with support for collections, textgroups, texts and passages browsing.

- The application will itself do the GetCapabilities request to retrieve the available texts and organize them through collection, textgroups and works.
- Once an edition or a translation is clicked, a page showing available references is shown.
- Once a passage is clicked, the passage is shown with available metadata.

```
# We import Flask
from flask import Flask
# We import Nemo
from flask.ext.nemo import Nemo
# We create an application. You can simply use your own
app = Flask(
    "My Application"
)
# We register a Nemo object with the minimal settings
nemo = Nemo(
    # API URL is the URL of your endpoint.
    api_url="http://services2.perseids.org/exist/restxq/cts",
    # We set up the base url to be empty. If you want nemo to be on a
    # subpath called "cts", you would have
    # base_url="cts",
    base_url="",
    # In our case, we have an inventory named "nemo"
    inventory="nemo",
    # We give thee ap object
    app=app
)
# We register its routes
nemo.register_routes()
# We register its filters
nemo.register_filters()
# We run the application
app.run()
```

Note: You can run this example using *python example.py default*

2.1.2 XSLT, CSS and Javascript addons

User Story

A developer, with no or only limited understanding of python, wants to expose their CTS works but have some modifications to do regarding the design.

Because Python is not a natural language and because not everybody knows it in academia, you might find yourself in a situation where you don't know it. On the other hand, XML TEI, HTML, CSS - and thus xsl and sometimes Javascript - are quite common languages known to both researchers and engineers. Capitains Nemo for Flask accepts custom templates, CSS, Javascript, XSL and statics. And in a simple, nice way :

```
# ...
nemo = Nemo(
    # Required API informations
    api_url="http://services2.perseids.org/exist/restxq/cts",
    base_url="",
```

```
    inventory="ciham",
    # For transform parameters, we provide a path to an xsl which will be used for every
    transform={"default" : "examples/ciham.xslt"},
    # For urntransform parameters, we provide a function which will be used to transform the urn for
    # this example just adds explanatory text
    urntransform={"default" : lambda urn: "Stable URI:" + str(urn)},
    # CSS value should be a list of path to CSS own files
    css=[
        "examples/ciham.css"
    ],
    # JS follows the same scheme
    js=[
        # use own js file to load a script to go from normalized edition to diplomatic one.
        "examples/ciham.js"
    ],
    templates={
        "menu": "examples/ciham.menu.html"
    },
    additional_static=[
        "path/to/picture.png"
    ]
)
```

Additional CSS, JS or Statics in Templates

To call or make a link to a static in your own template, you should always use the helper *url_for* and the route name *secondary_assets*. Additional statics can be linked to using the filename (be sure they do not collide !) and the type : css, js or static. Example : *{{url_for('nemo.secondary_assets', type='static', asset='picture.png')}}*.

Note: Templates are written with Jinja2. See also Templates.documentation. For XSL, we have some unfortunate restrictions, see *strip-spaces*

Note: You can run an example using css, js, templates and transform with *python example.py ciham*

2.1.3 Own Chunker

Warning: Starting from this example, the configuration and changes implied require the capacity to develop in Python.

User Story

> A developer wants to add a custom scheme for browsing text passages by groups that are not part of the citation scheme of the text. The custom scheme should be triggered by text identifier or using available CTS metadata about the text, such as the Citation Scheme.

CTS is good, but getValidReff can really be a hassle. The default generation of browsing level will always retrieve the deepest level of citations available. For the Iliad of Homer, which is composed of two levels, books and lines, this would translate to a GetValidReff level 2. This would mean that the generic chunker would return on the text page a link to each line of each book (it's a total of 15337 lines, if you did not know).
Chunker provides a simple, easy to develop interface to deal with such a situation : for example, returning only 50 lines groups of links (1.1-1.50, 1.51-1.100, etc.). The Nemo class accepts a chunker dictionary where **keys** are **urns** and where the key "**default**" is the default chunker to be applied. Given a chunker named *homer_chunker* and one named *default_chunker*, if the urn of Homer is **urn:cts:greekLit:tlg0012.tlg001.opp-grc1** (See Nemo.chunker.skeleton for function skeleton):

```
# ...
nemo = Nemo(
    # ...
    chunker= {
        "urn:cts:greekLit:tlg0012.tlg001.opp-grc1" : homer_chunker,
        "default": default_chunker
    }
   )
```

Note: You can run an example using chunker with *python example.py chunker*

Note: Parameters XSLT and prevnext work the same way. See relevant documentation : Nemo.chunker for more information about and examples of chunkers

2.1.4 Adding routes

User story

The user has needs in terms of new routes that would cover specific needs, like vis-a-vis edition.

There is multiple way to deal with this kind of situation. The best way is to create a subclass of Nemo. The idea behind that is that you rely on specific functionalities of Nemo and its context object. To deal with that and make as much as possible a good use of Nemo extension, you just need to add a new route to url using a tuple : first value would be the route, according to Flask standards, *ie /read/<collection>/<textgroup>/<work>/<version>/<passage_identifier>/<visavis>* , the name of the function or method (naming convention makes them start by r_), *ie r_double*, and a list of methods, by default ["GET"].

As you will most likely use a new template, don't forget to register it with the templates parameter !

```
# #We create a class based on Nemo
class NemoDouble(Nemo):
    def r_double(self, collection, textgroup, work, version, passage_identifier, visavis):
        """ Optional route to add a visavis version

        :param collection: Collection identifier
```

```
        :type collection: str
        :param textgroup: Textgroup Identifier
        :type textgroup: str
        :param work: Work identifier
        :type work: str
        :param version: Version identifier
        :type version: str
        :param passage_identifier: Reference identifier
        :type passage_identifier: str
        :param version: Visavis version identifier
        :type version: str
        :return: Template, version inventory object and Markup object representing the text
        :rtype: {str: Any}

        .. todo:: Change text_passage to keep being lxml and make so self.render turn etree element t
        """

        # Simply call the url of the
        args = self.r_passage(collection, textgroup, work, version, passage_identifier)
        # Call with other identifiers and add "visavis_" front of the argument
        args.update({ "visavis_{0}".format(key):value for key, value in self.r_passage(collection, te
        args["template"] = self.templates["r_double"]
        return args

nemo = NemoDouble(
    api_url="http://services2.perseids.org/exist/restxq/cts",
    base_url="",
    inventory="nemo",
    # We reuse Nemo.Routes and add a new one
    urls= Nemo.ROUTES + [("/read/<collection>/<textgroup>/<work>/<version>/<passage_identifier>/<visa
    css=[
        "examples/translations.css"
    ],
    # We think about registering the new route
    templates={
        "r_double": "./examples/translations/r_double.html"
    }
)
```

Note: You can run an example using chunker with *python example.py translations*

2.1.5 Replacing routes

2.2 Nemo Developper Guide

2.2.1 How to contribute ? Our github Etiquette

2.2.2 Writing and running tests

Running tests

Nemo is built for Python 3.4 and further.

2.2.3 Writing and building documentations

2.2.4 Coding guidelines

2.3 Templates documentation

2.3.1 Template dictionary

To replace all templates, just change your templates folder with the same filenames. If you wish to replace only one template, you need to use those keys :

```
TEMPLATES = {
    "container": "container.html",
    "menu": "menu.html",
    "text": "text.html",
    "textgroups": "textgroups.html",
    "index": "index.html",
    "texts": "texts.html",
    "version": "version.html",
    "passage_footer": "passage_footer.html"
}
```

2.3.2 Variables shared across templates

Variable Name	Details
assets['css']	List of css files to link to.
assets['js']	List of js files to link to.
url[*]	Dictionary where keys and values are derived from routes
templates[*]	Dictionary of templates with at least menu and container
collections	Collections list
lang	Lang to display

2.3.3 index.html

Only *Variables shared across templates*

2.3.4 menu.html

Only *Variables shared across templates*

2.3.5 textgroups.html

See *r_collection* in Nemo.api.r_collection

Variable Name	Details
textgroups	List of textgroups according to a collection

2.3.6 texts.html

See *r_texts* in Nemo.api.r_texts

Variable Name	Details
texts	List of texts according to a textgroup

2.3.7 version.html

See *r_version* in Nemo.api.r_version

Variable Name	Details
version	Version object with metadata about current text
reffs	List of tuples where first element is a reference, second a human readable translation

2.3.8 text.html

See *r_passage* in Nemo.api.r_passage

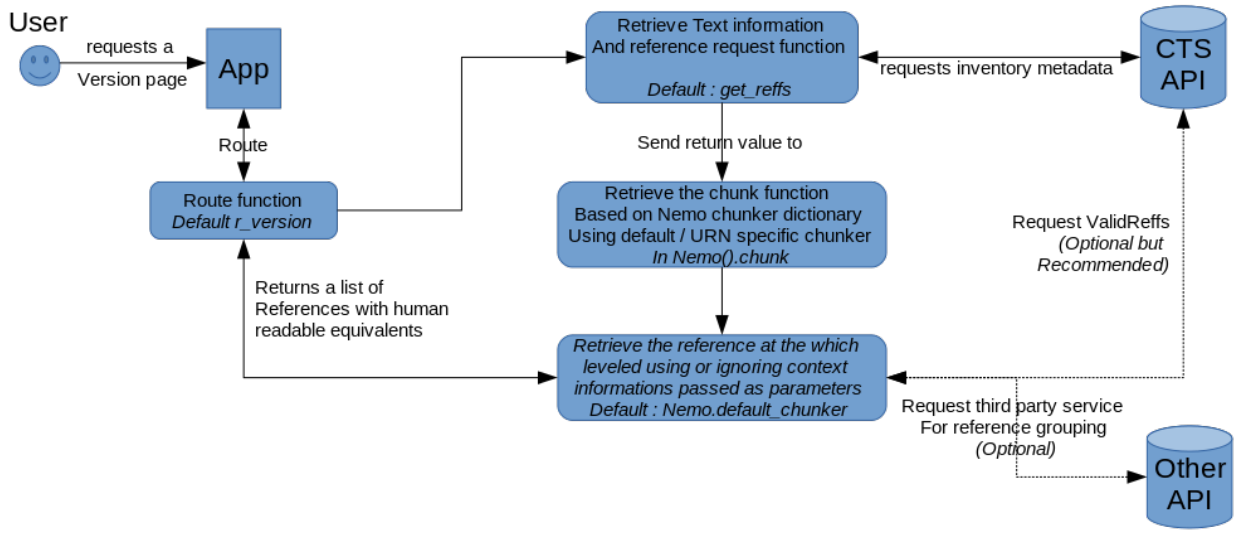
2.3.9 passage_footer.html

See *r_passage* in Nemo.api.r_passage

2.4 Chunkers, Transformers and GetPrevNext

Chunker, Transformers and GetPrevNext are way to customize your users' experience. Chunkers will decide what are the possible passages to see for a text, GetPrevNext what should be the previous and next passage while Transformers is a way to customize transformation, with or without XSLT

2.4.1 Process description



User story

A user browses available texts and select a text. He does not want a specific passages. Nemo proposes a list of passages based on the structure of the text.
Example: The Epigrams of Martial are a group of many books, each containing hundreds of poems, which are themselves composed of up to 50 lines. The use would preferably be proposed the poem as the minimal citation scheme for browsing, rather than each line.

To propose passages to the user, Capitains Nemo uses a chunker function which will group, if needed, references together. The function is called upon returning the list of references to the view. The function should always return a list of references, and not full urn, with a human readable version of it, which can be the same.

2.4.2 Chunkers

In the Nemo class, you'll find static methods called chunker, which can be used in the context of the Nemo().chunker dictionary. Chunkers are used to take care of grouping references when a user arrives on the version page of a text, to select where they should go.

Nemo contains multiple chunkers and accepts any contributions which provide helpful, transproject functions.

2.4.3 Defining a chunker in your Nemo implementation instance

The Nemo class accepts a chunker named argument that should be a dictionary where values are chunker functions. This dictionary should at least contain one key named "default". Any other key should represents a URN and will override the default function if the requested version has the given urn.

```
from flask.ext.nemo import Nemo
nemo = Nemo(chunker={
    "default": Nemo.default_chunker,
    "urn:cts:latinLit:phi1294.phi002.perseus-lat2": Nemo.scheme_chunker,
    # This will override the original function and provides a poem based reference for Martial Epigra
    "urn:cts:latinLit:phi1017.phi004.opp-lat4": lambda version, callback: Nemo.line_chunker(version,
    # Use a lambda to override default line numbers returned by Nemo.line_chunker for Seneca's Medea
})
```

Note: See Nemo.api documentation

Building your own : Structure, Parameters, Return Values

```
# Chunker skeleton
def chunker_name(version, getValidReff):
    """ Document what your chunker should do

    :param version: A version object according to MyCapytains standards. It contains metadata about t
    :type version: MyCapytains.resources.inventory.Text
    :param getValidReff: Callback function to perform a getValidReff on the given param. It accepts a
    :type getValidReff: function(level) -> [str]
    :return: A list of tuple of strings where the first element is the CTS URN reference part and the
    :rtype: [(str, str)]
    """
    return [("1.pr", "Book 1 Prolog")("1.1", "Book 1 Poem 1"), ...]
```

A chunker should take always at least two positional arguments :

- The first one will be the version, based on a MyCapytains.resources.inventory.Text class. It contains information about the citation scheme for example.
- The second one is a callback function that the chunker can use to retrieve the valid references. This callback itself takes a parameter named level. This callback corresponds to a MyCapytains.resources.texts.api.getValidReff() method. It returns a list of string based urns.

The chunker itself should return a list of tuples where the first element is a passage reference such as "1.pr" or "1-50" and a second value which is a readable version of this citation node.

Note: As seen in the diagram, there is no limitation for the chunker as long as it returns a valid list of references and their human readable version. It could in theory ask a third party service to return page-based urns to browse a text by pages according its OCR source / manuscript

---

```
# Example of chunker for the Satura of Juvenal
def satura_chunker(version, getValidReff):
    reffs = [urn.split(":")[-1] for urn in getValidReff(level=2)]
    # Satura scheme contains three level (book, poem, lines) but only the Satura number is sequentia
    # So as human readable, we give only the second member of the reference body
    return [(reff, "Satura {0}".format(reff.split(".")[-1])) for reff in reffs]1
```

Available chunkers

static Nemo.**default_chunker**(*text*, *getreffs*)

This is the default chunker which will resolve the reference giving a callback (getreffs) and a text object with its metadata

Parameters

- **text** (*MyCapytains.resources.inventory.Text*) – Text Object representing either an edition or a translation
- **getreffs** (*function*) – callback function which retrieves a list of references

Returns List of urn references with their human readable version

Return type [(str, str)]

static Nemo.**line_chunker**(*text*, *getreffs*, *lines=30*)

Groups line reference together

Parameters

- **text** (*MyCapytains.resources.text.api*) – Text object
- **getreffs** (*function(level)*) – Callback function to retrieve text
- **lines** (*int*) – Number of lines to use by group

Returns List of grouped urn references with their human readable version

Return type [(str, str)]

static Nemo.**scheme_chunker**(*text*, *getreffs*)

This is the scheme chunker which will resolve the reference giving a callback (getreffs) and a text object with its metadata

Parameters

- **text** (*MyCapytains.resources.inventory.Text*) – Text Object representing either an edition or a translation
- **getreffs** (*function*) – callback function which retrieves a list of references

Returns List of urn references with their human readable version

Return type [(str, str)]

static Nemo.**level_chunker**(*text*, *getValidReff*, *level=1*)

Chunk a text at the passage level

Parameters

- **text** (*MyCapytains.resources.text.api*) – Text object
- **getreffs** (*function(level)*) – Callback function to retrieve text

Returns List of urn references with their human readable version

Return type [(str, str)]

2.4.4 PrevNext

PrevNext follows the same scheme as Chunker.

2.4.5 Transformers

Transformers should always return a string

2.5 Restriction regarding XSLT

2.5.1 strip-spaces

Strip spaces does not work due to a bug found in the C library backing up Python LXML (lib-xslt). The bug, referenced here https://bugzilla.gnome.org/show_bug.cgi?id=620102 , is known but not tackled for the moment. Here is a work around :

```
<xsl:template match="text()">
   <xsl:if test="not(normalize-space()='')"><xsl:copy/></xsl:if>
</xsl:template>
```

2.6 Nemo API

class flask.ext.nemo.**Nemo**(*name=None*, *app=None*, *api_url='/'*, *retriever=None*, *base_url='/nemo'*, *cache=None*, *expire=3600*, *template_folder=None*, *static_folder=None*, *static_url_path=None*, *urls=None*, *inventory=None*, *transform=None*, *urn-transform=None*, *chunker=None*, *prevnext=None*, *css=None*, *js=None*, *templates=None*, *statics=None*)

Nemo is an extension for Flask python micro-framework which provides a User Interface to your app for dealing with CTS API.

Parameters

- **app** (*Flask*) – Flask application

- **api_url** (*str*) – URL of the API Endpoint
- **retriever** (*MyCapytain.retrievers.proto.CTS*) – CTS Retriever (Will be defaulted to api_url using cts5 retriever if necessary)
- **base_url** (*str*) – Base URL to use when registering the endpoint
- **cache** – SQLITE cache file name
- **expire** – TIme before expiration of cache, default 3600
- **template_folder** (*str*) – Folder in which the templates can be found
- **static_folder** (*str*) – Folder in which statics file can be found
- **static_url_path** (*str*) – Base url to use for assets
- **urls** (*[(str, str, [str])]*) – Function and routes to register (See Nemo.ROUTES)
- **inventory** (*str*) – Default inventory to use
- **transform** (*bool|dict*) – Dictionary of XSL filepath or transform function where default key is the default applied function
- **urntransform** (*bool|dict*) – Dictionary of urn transform functions where default key is the default applied function
- **chunker** (*{str: function(str, function(int))}*) – Dictionary of function to group responses of GetValidReff
- **prevnext** (*{str: function(str, function())}*) – Dictionary of function to execute GetPrevNext
- **css** (*[str]*) – Path to additional stylesheets to load
- **js** (*[str]*) – Path to additional javascripts to load
- **templates** (*{str: str}*) – Register or override templates (Dictionary of index / path)
- **statics** (*[str]*) – Path to additional statics such as picture to load

Warning: Until a C libxslt error is fixed (https://bugzilla.gnome.org/show_bug.cgi?id=620102), it is not possible to use strip spaces in the xslt given to this application. See *strip-spaces*

2.6.1 Flask related function

Nemo.**init_app**(*app*)
Initiate the application

Parameters **app** (*flask.Flask*) – Flask application on which to add the extension

Nemo.**register_routes**()
Register routes on app using Blueprint

Returns Nemo blueprint

Return type flask.Blueprint

Nemo.**register_filters**()
Register filters for Jinja to use

Note: Extends the dictionary filters of jinja_env using self._filters list

Nemo.**create_blueprint**()
Create blueprint and register rules

Returns Blueprint of the current nemo app

Return type flask.Blueprint

2.6.2 Controller

Specific methods

Nemo.**get_inventory**()
Request the api endpoint to retrieve information about the inventory

Returns The text inventory

Return type MyCapytain.resources.inventory.TextInventory

Nemo.**get_collections**()
Filter inventory and make a list of available collections

Returns A set of CTS Namespaces

Return type set(str)

Nemo.**get_textgroups**(*collection_urn=None*)
Retrieve textgroups

Parameters **collection_urn** (*str*) – Collection to use for filtering the textgroups

Returns List of textgroup filtered by collection

Return type [MyCapytain.resources.inventory.Textgroup]

Nemo.**get_works**(*collection_urn=None*, *textgroup_urn=None*)
Retrieve works

Parameters

- **collection_urn** (*str*) – Collection to use for filtering the textgroups
- **textgroup_urn** (*str*) – Textgroup to use for filtering the works

Returns List of work filtered by collection/Textgroup

Return type [MyCapytain.resources.inventory.Work]

Nemo.**get_texts**(*collection_urn=None*, *textgroup_urn=None*, *work_urn=None*)
Retrieve texts

Parameters

- **collection_urn** (*str*) – Collection to use for filtering the textgroups
- **textgroup_urn** (*str*) – Textgroup to use for filtering the works
- **work_urn** (*str*) – Work to use for filtering the texts

Returns List of texts filtered by parameters

Return type [MyCapytain.resources.inventory.Text]

Nemo.**get_text**(*collection_urn*, *textgroup_urn*, *work_urn*, *version_urn*)
Retrieve one version of a Text

Parameters

- **collection_urn** (*str*) – Collection to use for filtering the textgroups
- **textgroup_urn** (*str*) – Textgroup to use for filtering the works
- **work_urn** (*str*) – Work identifier to use for filtering texts
- **version_urn** (*str*) – Version identifier

Returns A Text represented by the various parameters

Return type MyCapytain.resources.inventory.Text

Nemo.**get_reffs**(*collection*, *textgroup*, *work*, *version*)
Get the setup for valid reffs.

Returns the inventory text object with its metadata and a callback function taking a level parameter and returning a list of strings.

Parameters

- **collection** (*str*) – Collection identifier
- **textgroup** (*str*) – Textgroup identifier
- **work** (*str*) – Work identifier
- **version** (*str*) – Version identifier

Returns Text with its metadata, callback function to retrieve validreffs

Return type (MyCapytains.resources.texts.api.Text, lambda: [str])

Nemo.**get_passage**(*collection*, *textgroup*, *work*, *version*, *passage_identifier*)
Retrieve the passage identified by the parameters

Parameters

- **collection** (*str*) – Collection identifier
- **textgroup** (*str*) – Textgroup identifier
- **work** (*str*) – Work identifier
- **version** (*str*) – Version identifier
- **passage_identifier** (*str*) – Reference Identifier

Returns A Passage object containing informations about the passage

Return type MyCapytain.resources.texts.api.Passage

Customization appliers

Nemo.**chunk**(*text*, *reffs*)
Handle a list of references depending on the text identifier using the chunker dictionary.

Parameters

- **text** (*MyCapytains.resources.texts.api.Text*) – Text object from which comes the references

- **reffs** (*callback(level)*) – Callback function to retrieve a list of string with a level parameter

Returns Transformed list of references

Return type [str]

Nemo.**getprevnext**(*passage*, *callback*)
Retrieve previous and next passage using

Parameters

- **text** (*MyCapytains.resources.texts.api.Passage*) – Text object from which comes the references
- **reffs** (*callback()*) – Callback function to retrieve a tuple where first element is the previous passage, and second the next

Returns Reference of previous passage, reference of next passage

Return type (str, str)

Nemo.**transform**(*work*, *xml*)
Transform input according to potentiallyregistered XSLT

Note: Due to XSLT not being able to be used twice, we rexsltise the xml at every call of xslt

Warning: Until a C libxslt error is fixed (https://bugzilla.gnome.org/show_bug.cgi?id=620102), it is not possible to use strip tags in the xslt given to this application

Parameters

- **work** (*MyCapytains.resources.inventory.Text*) – Work object containing metadata about the xml
- **xml** (*etree._Element*) – XML to transform

Returns String representation of transformed resource

Return type str

Nemo.**transform_urn**(*urn*)
Transform urn according to configurable function

Parameters **urn** (*URN*) – URN to transform

Returns the URN (transformed or not)

Return type URN

Shared methods

Nemo.**render**(*template*, ***kwargs*)
Render a route template and adds information to this route.

Parameters

- **template** (*str*) – Template name
- **kwargs** (*dict*) – dictionary of named arguments used to be passed to the template

Returns Http Response with rendered template

Return type flask.Response

Nemo.**make_breadcrumbs**(***kwargs*)
Make breadcrumbs for a route

Parameters **kwargs** (*dict*) – dictionary of named arguments used to construct the view

Returns List of dict items the view can use to construct the link.

Return type list({ "link": str, "title", str, "args", dict})

Nemo.**view_maker**(*name*)
Create a view

Parameters **name** (*str*) – Name of the route function to use for the view.

Returns Route function which makes use of Nemo context (such as menu informations)

Return type function

Nemo.**route**(*fn*, ***kwargs*)
Route helper : apply fn function but keep the calling object, *ie* kwargs, for other functions

Parameters

- **fn** (*function*) – Function to run the route with
- **kwargs** (*dict*) – Parsed url arguments

Returns HTTP Response with rendered template

Return type flask.Response

2.6.3 Routes

Nemo.**r_index**()
Homepage route function

Returns Template to use for Home page

Return type {str: str}

Nemo.**r_collection**(*collection*)
Collection content browsing route function

Parameters **collection** (*str*) – Collection identifier

Returns Template and textgroups contained in given collections

Return type {str: Any}

Nemo.**r_texts**(*collection*, *textgroup*)
Textgroup content browsing route function

Parameters

- **collection** (*str*) – Collection identifier
- **textgroup** (*str*) – Textgroup Identifier

Returns Template and texts contained in given textgroup

Return type {str: Any}

Nemo.**r_version**(*collection*, *textgroup*, *work*, *version*)
Text exemplar references browsing route function

Parameters

- **collection** (*str*) – Collection identifier
- **textgroup** (*str*) – Textgroup Identifier
- **work** (*str*) – Work identifier
- **version** (*str*) – Version identifier

Returns Template, version inventory object and references urn parts

Return type

{ "template" : str, "version": MyCapytains.resources.inventory.Text, "reffs": [str] }

Nemo.**r_passage**(*collection*, *textgroup*, *work*, *version*, *passage_identifier*)

Retrieve the text of the passage

Parameters

- **collection** (*str*) – Collection identifier
- **textgroup** (*str*) – Textgroup Identifier
- **work** (*str*) – Work identifier
- **version** (*str*) – Version identifier
- **passage_identifier** (*str*) – Reference identifier

Returns Template, version inventory object and Markup object representing the text

Return type {str: Any}

Nemo.**r_assets**(*type*, *asset*)

Route for specific assets.

Parameters **asset** – Filename of an asset

Returns Response

2.6.4 Statics

Filters

Filters follow a naming convention : they should always start with "**f_**"

static Nemo.**f_active_link**(*string*, *url*)

Check if current string is in the list of names

Parameters **string** – String to check for in url

Returns CSS class "active" if valid

Return type str

static Nemo.**f_collection_i18n**(*string*)

Return a i18n human readable version of a CTS domain such as latinLit

Parameters **string** (*str*) – CTS Domain identifier

Returns Human i18n readable version of the CTS Domain

Return type str

static Nemo.**f_formatting_passage_reference**(*string*)
Get the first part only of a two parts reference

Parameters **string** (*str*) – A urn reference part

Returns First part only of the two parts reference

Return type str

static Nemo.**f_order_text_edition_translation**(*versions_list*)
Takes a list of versions and put translations after editions

Parameters **versions_list** (*[Text]*) – List of text versions

Returns List where first members will be editions

Return type [Text]

static Nemo.**f_i18n_citation_type**(*string*, *lang='eng'*)
Take a string of form %citation_type|passage% and format it for human

Parameters

- **string** – String of formation %citation_type|passage%
- **lang** – Language to translate to

Returns Human Readable string

Helpers

static Nemo.**map_urns**(*items*, *query*, *part_of_urn=1*, *attr='textgroups'*)
Small function to map urns to filter out a list of items or on a parent item

Parameters

- **items** (*MyCapytains.resources.inventory.Resource*) – Inventory object
- **query** (*str*) – Part of urn to check against
- **part_of_urn** (*int*) – Identifier of the part of the urn

Returns Items corresponding to the object children filtered by the query

Return type list(items.children)

static Nemo.**filter_urn**(*item*, *part_of_urn*, *query*)
Small function to map urns to filter out a list of items or on a parent item

Parameters

- **item** – Inventory object
- **query** (*str*) – Part of urn to check against
- **part_of_urn** (*int*) – Identifier of the part of the urn

Returns Items corresponding to the object children filtered by the query

Return type list(items.children)

static Nemo.**in_and_not_in**(*identifier*, *collection*, *kwargs*)
Check if an element identified by identifier is in kwargs but not the collection containing it

Parameters

- **identifier** (*str*) – URL Identifier of one kind of element (Textgroup, work, etc.)

- **collection** (*str*) – Resource identifier of one kind of element (Textgroup, work, etc.)
- **kwargs** (*{str: Any}*) – Arguments passed to a template

Returns Indicator of presence of required informations

Return type bool

static Nemo.**prevnext_callback_generator**(*passage*)
Default callback generator to retrieve prev and next value of a passage

Parameters **passage** (*MyCapytains.resources.texts.api.Passage*) – Passage for which to get previous and following reference

Returns Function to retrieve those information

Return type function

Chunkers

static Nemo.**default_chunker**(*text*, *getreffs*)
This is the default chunker which will resolve the reference giving a callback (getreffs) and a text object with its metadata

Parameters

- **text** (*MyCapytains.resources.inventory.Text*) – Text Object representing either an edition or a translation
- **getreffs** (*function*) – callback function which retrieves a list of references

Returns List of urn references with their human readable version

Return type [(str, str)]

static Nemo.**line_chunker**(*text*, *getreffs*, *lines=30*)
Groups line reference together

Parameters

- **text** (*MyCapytains.resources.text.api*) – Text object
- **getreffs** (*function(level)*) – Callback function to retrieve text
- **lines** (*int*) – Number of lines to use by group

Returns List of grouped urn references with their human readable version

Return type [(str, str)]

static Nemo.**scheme_chunker**(*text*, *getreffs*)
This is the scheme chunker which will resolve the reference giving a callback (getreffs) and a text object with its metadata

Parameters

- **text** (*MyCapytains.resources.inventory.Text*) – Text Object representing either an edition or a translation
- **getreffs** (*function*) – callback function which retrieves a list of references

Returns List of urn references with their human readable version

Return type [(str, str)]

PrevNexter

static Nemo.**default_prevnext**(*passage*, *callback*)
Default deliver of prevnext informations

Parameters

- **passage** (*MyCapytains.resources.texts.api.Passage*) – Passage for which to get previous and following reference
- **callback** (*function*) – Function to retrieve those information

Returns Tuple representing previous and following reference

Return type (str, str)