MyCapytains Documentation

Release 0.0.1

Thibault Clérice

December 06, 2016

MyCapytain is a python package which provides a large set of tools to deal with CTS and Capitains Guidelines in Python.

CHAPTER 1

Installation and Requirements

The best way to install MyCapytain is to use pip. MyCapytain tries to support both Python 2.7 and Python 3.4.

```
pip install MyCapytain
```

If you prefer to use setup.py, you should clone and use the following

```
git clone https://github.com/Capitains/MyCapytain.git
cd MyCapytain
python setup.py install
```

CHAPTER 2

Contents

2.1 Project using MyCapytain

If you are using MyCapytain and wish to appear here, please feel free to open an issue

2.1.1 Extensions

Nautilus

Nautilus provides a local retriever to build inventory based on a set of folders available locally.

Flask Capitains Nemo

Flask Capitains Nemo is an extension for Flask to build a browsing interface using both retrievers and resources modules. You will find example of use in a web based environment.

HookTest

HookTest is a library and command line tools for checking resources against the Capitains Guidelines You'll find uses mainly in units.py

2.2 The MyCapytain local file implementation

2.2.1 Introduction

The module *MyCapytain.resources.local.text* requires the guidelines of Capitains to be implemented in your files.

2.2.2 Basics and examples

Getting all passages from a text

```python
# We import the correct classes from the local module
from MyCapytain.resources.texts.local import Text, Passage

# We open a file
with open("/tests/testing_data/texts/sample.xml") as f:
    # We initiate a Text object giving the IO instance to resource argument
    text = Text(resource=f)

# Text objects have a citation property
# len(Citation(...)) gives the depth of the citation scheme
# in the case of this sample, this would be 3 (Book, Poem, Line)
for ref in text.getValidReff(level=len(text.citation)):
    # We retrieve a Passage object for each reference that we find
    # We can pass the reference many way, including in the form of a list of strings
    psg = text.getPassage(ref.split("."), hypercontext=False)
    # We print the passage from which we retrieve <note> nodes
    print("\t".join([ref, psg.text(exclude=["note"])]))
```

2.3 Endpoints

2.3.1 Introduction

A first important point for MyCapytains endpoint is that the resources are not parsed into object but should only provide the request by default.

2.3.2 Getting Passage from an endpoint using a Retriever

```python
from MyCapytain.retrievers import cts5
from MyCapytain.resources.texts.api import Text

# We set the variable up, as if we were in a function
# This URN won't work (urn:cts:greekLit:tlg0032.tlg005.perseus-grc1) because it has no TEI namespace
urn = 'urn:cts:latinLit:phi1294.phi002.perseus-lat2'
ref = "1.1-1.2"

# We set the api up. Endpoint takes one required argument
#  (the URI) and one inventory as optional argument
cts = cts5.CTS('http://services2.perseids.org/exist/restxq/cts', inventory="nemo")

# We set up a text object to be able to retrieve passage of it
# Text in API modules takes endpoint as resource and URN as param

text = Text(urn=urn, resource=cts)

# We use the method getPassage which takes a reference argument
passage = text.getPassage(reference=ref)

# Passage then has different methods and properties
# Most of them (except next, prev and prevnext properties) are inherited from MyCapytain.resources.te
# For example
print(passage.text(exclude=["note", "head"]))  # Will get the text without "note" and "head" TEI node
print(passage.xml)  # The xml property can be used as an argument for XSLT for example
```

2.4 Known issues and recommendations

2.4.1 XPath Issues

lxml, which is the package powering xml support here, does not accept XPath notations such as */div/(a or b)[@n]*. Solution for this edge case is */div/*[self::a or self::b][@n]*

2.5 MyCapytain API Documentation

2.5.1 Utilities, metadata and references

Module common contains tools such as a namespace dictionary as well as cross-implementation objects, like URN, Citations...

URN, References and Citations

class `MyCapytain.common.reference.`**`URN`**(*urn*)
A URN object giving all useful sections

Parameters **`urn`** (*`str`*) – A CTS URN

Variables

- **`urn_namespace`** – Namespace of the URN
- **`namespace`** – CTS Namespace
- **`textgroup`** – CTS Textgroup
- **`work`** – CTS Work
- **`version`** – CTS Version
- ***`reference`*** – CTS Reference
- **`NAMESPACE`** – Constant representing the URN until its namespace
- **`TEXTGROUP`** – Constant representing the URN until its textgroup
- **`WORK`** – Constant representing the URN until its work
- **`VERSION`** – Constant representing the URN until its version
- **`PASSAGE`** – Constant representing the URN until its full passage
- **`PASSAGE_START`** – Constant representing the URN until its passage (end excluded)
- **`PASSAGE_END`** – Constant representing the URN until its passage (start excluded)
- **`NO_PASSAGE`** – Constant representing the URN until its passage excluding its passage
- **`COMPLETE`** – Constant representing the complete URN

Example

```
>>>    a = URN(urn="urn:cts:latinLit:phi1294.phi002.perseus-lat2:1.1")
```

URN object supports the following magic methods : len(), str() and eq(), gt() and lt().

Example

```
>>>     b = URN("urn:cts:latinLit:phi1294.phi002")
>>>     a != b
>>>     a > b # It has more member. Only member count is compared
>>>     b < a
>>>     len(a) == 5 # Reference is not counted to not induce count equivalencies with th
>>>     len(b) == 4
```

upTo(*key*)

Returns the urn up to given level using URN Constants

Parameters **key** (*int*) – Identifier of the wished resource using URN constants

Returns String representation of the partial URN requested

Return type str

Example

```
>>>     a = URN(urn="urn:cts:latinLit:phi1294.phi002.perseus-lat2:1.1")
>>>     a.upTo(URN.TEXTGROUP) == "urn:cts:latinLit:phi1294"
```

class MyCapytain.common.reference.**Reference**(*reference=u''*)

A reference object giving informations

Parameters **reference** (*basestring*) – Passage Reference part of a Urn

Variables

- ***parent*** – Parent Reference
- ***highest*** – List representation of the range member which is the highest in the hierarchy (If equal, start is returned)
- ***start*** – First part of the range
- ***end*** – Second part of the range
- ***list*** – List representation of the range. Not available for range
- ***subreference*** – Word and Word counter (“Achiles”, 1) representing the subreference. Not available for range

Example

```
>>>     a = Reference(reference="1.1@Achiles[1]-1.2@Zeus[1]")
>>>     b = Reference(reference="1.1")
>>>     Reference("1.1-2.2.2").highest == ["1", "1"]
```

Reference object supports the following magic methods : len(), str() and eq().

Example

```
>>>     len(a) == 2 && len(b) == 1
>>>     str(a) == "1.1@Achiles[1]-1.2@Zeus[1]"
>>>     b == Reference("1.1") && b != a
```

Note: While Reference(...).subreference and .list are not available for range, Reference(..).start.subreference and Reference(..).end.subreference as well as .list are available

end
Quick access property for reference end list

highest
Return highest reference level

For references such as 1.1-1.2.8, with different level, it can be useful to access to the highest node in the hierarchy. In this case, the highest level would be 1.1. The function would return [”1”, “1”]

Note: By default, this property returns the start level

Return type *Reference*

list
Return a list version of the object if it is a single passage

Note: Access to start list and end list should be done through obj.start.list and obj.end.list

Return type [str]

parent
Parent of the actual URN, for example, 1.1 for 1.1.1

Return type *Reference*

start
Quick access property for start list

subreference
Return the subreference of a single node reference

Note: Access to start and end subreference should be done through obj.start.subreference

and obj.end.subreference

Return type (str, int)

class `MyCapytain.common.reference.`**`Citation`**(*name=None*, *xpath=None*, *scope=None*, *refsDecl=None*, *child=None*)
A citation object gives informations about the scheme

Parameters

- **name** (*basestring*) – Name of the citation (e.g. “book”)
- **xpath** (*basestring*) – Xpath of the citation (As described by CTS norm)
- **scope** – Scope of the citation (As described by CTS norm)
- **refsDecl** (*basestring*) – refsDecl version
- **child** (`Citation`) – A citation

Variables

- **name** – Name of the citation (e.g. “book”)
- ***xpath*** – Xpath of the citation (As described by CTS norm)

- **scope** – Scope of the citation (As described by CTS norm)
- **refsDecl** – refsDecl version
- **child** – A citation

__iter__()
Iteration method

Loop over the citation childs

Example

```
>>>    c = Citation(name="line")
>>>    b = Citation(name="poem", child=c)
>>>    a = Citation(name="book", child=b)
>>>    [e for e in a] == [a, b, c]
```

__len__()
Length method

Return type int

Returns Number of nested citations

fill(*passage=None*, *xpath=None*)
Fill the xpath with given informations

Parameters

- **passage** (*Reference or list or None. Can be list of None and not None*) – Passage reference
- **xpath** (*Boolean*) – If set to True, will return the replaced self.xpath value and not the whole self.refsDecl

Return type basestring

Returns Xpath to find the passage

```
citation = Citation(name="line", scope="/TEI/text/body/div/div[@n="?"]",xpath="//l[@n="?"]")
print(citation.fill(["1", None]))
# /TEI/text/body/div/div[@n='1']//l[@n]
print(citation.fill(None))
# /TEI/text/body/div/div[@n]//l[@n]
print(citation.fill(Reference("1.1"))
# /TEI/text/body/div/div[@n='1']//l[@n='1']
print(citation.fill("1", xpath=True)
# //l[@n='1']
```

Metadata containers

class MyCapytain.common.metadata.**Metadata**(*keys=None*)
Bases: `future.types.newobject.newobject`

A metadatum aggregation object provided to centralize metadata

Parameters key (*List.<basestring>*) – A metadata field name

Variables ***metadata*** – Dictionary of metadatum

__getitem__(*key*)
Add a quick access system through getitem on the instance

Parameters **key** (*basestring, int, tuple*) – Index key representing a set of metadatum

Returns An element of children whose index is key

Raises *KeyError* If key is not registered or recognized

Example

```
>>>    a = Metadata()
>>>    m1 = Metadatum(name="title", [("lat", "Amores"), ("fre", "Les Amours")])
>>>    m2 = Metadatum(name="author", [("lat", "Ovidius"), ("fre", "Ovide")])
>>>    a[("title", "author")] = (m1, m2)
```

```
>>>    a["title"] == m1
>>>    a[0] == m1
>>>    a[("title", "author")] == (m1, m2)
```

__setitem__(*key*, *value*)
Set a new metadata field

Parameters

- **key** (*basestring, tuple*) – Name of metadatum field
- **value** (Metadatum) – Metadum dictionary

Returns An element of children whose index is key

Raises *TypeError* if key is not basestring or tuple of basestring

Raises *ValueError* if key and value are list and are not the same size

Example

```
>>>    a = Metadata()
```

```
>>>    a["title"] = Metadatum(name="title", [("lat", "Amores"), ("fre", "Les Amours")
>>>    print(a["title"]["lat"]) # Amores
```

```
>>>    a[("title", "author")] = (
>>>         Metadatum(name="title", [("lat", "Amores"), ("fre", "Les Amours")]),
>>>         Metadatum(name="author", [("lat", "Ovidius"), ("fre", "Ovide")])
>>>     )
>>>    print(a["title"]["lat"], a["author"]["fre"]) # Amores, Ovide
```

__iter__()
Iter method of Metadata

Example

```
>>> a = Metadata(("title", "desc", "author"))
>>> for key, value in a:
>>>     print(key, value) # Print ("title", "<Metadatum object>") then ("desc", "<Met
```

__len__()
Returns the number of Metadatum registered in the object

Return type int

Returns Number of metadatum objects

Example

```
>>>     a = Metadata(("title", "description", "author"))
>>>     print(len(a)) # 3
```

`__add__`(*other*)

Merge Metadata objects together

Parameters **`other`** (`Metadata`) – Metadata object to merge with the current one

Returns The merge result of both metadata object

Return type *Metadata*

Example

```
>>> a = Metadata(name="label")
>>> b = Metadata(name="title")
>>> a + b == Metadata(name=["label", "title"])
```

class `MyCapytain.common.metadata.`**`Metadatum`**(*name*, *children=None*)

Bases: `future.types.newobject.newobject`

Metadatum object represent a single field of metadata

Parameters

- **`name`** (*`basestring`*) – Name of the field
- **`children`** (*`List`*) – List of tuples, where first element is the key, and second the value

Example

```
>>>     a = Metadatum(name="label", [("lat", "Amores"), ("fre", "Les Amours")])
>>>     print(a["lat"]) # == "Amores"
```

`__getitem__`(*key*)

Add an iterable access method

Int typed key access to the *n* th registered key in the instance. If string based key does not exist, see for a default.

Parameters **`key`** (*`basestring, tuple, int`*) – Key of wished value

Returns An element of children whose index is key

Raises KeyError if key is unknown (when using Int based key or when default is not set)

Example

```
>>>     a = Metadatum(name="label", [("lat", "Amores"), ("fre", "Les Amours")])
>>>     print(a["lat"]) # Amores
>>>     print(a[("lat", "fre")]) # Amores, Les Amours
>>>     print(a[0]) # Amores
>>>     print(a["dut"]) # Amores
```

`__setitem__`(*key*, *value*)

Register index key and value for the instance

Parameters

- **`key`** (*`basestring, list, tuple`*) – Index key(s) for the metadata
- **`value`** (*`basestring, list, tuple`*) – Values for the metadata

Returns An element of children whose index is key

Raises *TypeError* if key is not basestring or tuple of basestring

Raises *ValueError* if key and value are list and are not the same size

Example

```
>>> a = Metadatum(name="label")
```

```
>>> a["eng"] = "Illiad"
>>> print(a["eng"]) # Illiad
```

```
>>> a[("fre", "grc")] = ("Illiade", "λι")
>>> print(a["fre"], a["grc"]) # Illiade, λι
```

```
>>> a[("ger", "dut")] = "Iliade"
>>> print(a["ger"], a["dut"]) # Iliade, Iliade
```

__iter__()

Iter method of Metadatum

Example

```
>>> a = Metadata(name="label", [("lat", "Amores"), ("fre", "Les Amours")])
>>> for key, value in a:
>>>     print(key, value) # Print ("lat", "Amores") and then ("fre", "Les Amours")
```

setDefault(*key*)

Set a default key when a field does not exist

Parameters **key** (`basestring`) – An existing key of the instance

Returns Default key

Raises *ValueError* If key is not registered

Example

```
>>>    a = Metadatum(name="label", [("lat", "Amores"), ("fre", "Les Amours")])
>>>    a.setDefault("fre")
>>>    print(a["eng"]) # == "Les Amours"
```

Utilities

`MyCapytain.common.utils.`**NS = {u'xml': u'http://www.w3.org/XML/1998/namespace', u'tei': u'http://www.tei-c.org/ns/1**

Dictionary of namespace that can be useful

class `MyCapytain.common.utils.`**OrderedDefaultDict**(*default_factory=None*, **args*, ***kwargs*)

Bases: `collections.OrderedDict`

`MyCapytain.common.utils.`**copyNode**(*node*, *children=False*, *parent=False*)

Parameters

- **node** –
- **children** –

- **`parent`** –

Returns

`MyCapytain.common.utils.`**`formatXpath`**(*xpath*)

Parameters **`xpath`** –

Returns

`MyCapytain.common.utils.`**`nested_get`**(*dictionary*, *keys*)
Get value in dictionary for dictionary[keys[0]][keys[1]][keys[..n]]

Parameters

- **`dictionary`** – An input dictionary
- **`keys`** – Keys where to store data

Returns

`MyCapytain.common.utils.`**`nested_ordered_dictionary`**()

`MyCapytain.common.utils.`**`nested_set`**(*dictionary*, *keys*, *value*)
Set value in dictionary for dictionary[keys[0]][keys[1]][keys[..n]]

Parameters

- **`dictionary`** – An input dictionary
- **`keys`** – Keys where to store data
- **`value`** – Value to set at keys** target

Returns None

`MyCapytain.common.utils.`**`normalize`**(*string*)
Remove double-or-more spaces in a string

Parameters **`string`** (*`basestring`*) – A string to change

Return type Basestring

Returns Clean string

`MyCapytain.common.utils.`**`normalizeXpath`**(*xpath*)
Normalize XPATH split around slashes

Parameters **`xpath`** (*`[str]`*) – List of xpath elements

Returns List of refined xpath

Return type [str]

`MyCapytain.common.utils.`**`passageLoop`**(*parent*, *new_tree*, *xpath1*, *xpath2=None*, *preceding_siblings=False*, *following_siblings=False*)
Loop over passages to construct and increment new tree given a parent and XPaths

Parameters

- **`parent`** – Parent on which to perform xpath
- **`new_tree`** – Parent on which to add nodes
- **`xpath1`** (*`[str]`*) – List of xpath elements
- **`xpath2`** (*`[str]`*) – List of xpath elements
- **`preceding_siblings`** – Append preceding siblings of XPath 1/2 match to the tree

- **following_siblings** – Append following siblings of XPath 1/2 match to the tree

Returns Newly incremented tree

`MyCapytain.common.utils.`**`performXpath`**(*parent*, *xpath*)
Perform an XPath on an element and indicate if we need to loop over it to find something

Parameters

- **parent** – XML Node on which to perform XPath
- **xpath** – XPath to run

Returns (Result, Need to loop Indicator)

`MyCapytain.common.utils.`**`xmliter`**(*node*)
Provides a simple XML Iter method which complies with either _Element or _ObjectifiedElement

Parameters **node** – XML Node

Returns Iterator for iterating over children of said node.

`MyCapytain.common.utils.`**`xmlparser`**(*xml*)
Parse xml

Parameters **xml** (*basestring, lxml.etree._Element*) – XML element

Return type lxml.etree._Element

Returns An element object

Raises TypeError if element is not in accepted type

2.5.2 API Retrievers

Module endpoints contains prototypes and implementation of retrievers in MyCapytain

Ahab

CTS 5 API

class `MyCapytain.retrievers.cts5.`**`CTS`**(*endpoint*, *inventory=None*)
Bases: *MyCapytain.retrievers.proto.CTS*

Basic integration of the MyCapytain.retrievers.proto.CTS abstraction

`call`(*parameters*)
Call an endpoint given the parameters

Parameters **parameters** (*dict*) – Dictionary of parameters

Return type *text*

`getCapabilities`(*inventory=None*)
Retrieve the inventory information of an API

Parameters **inventory** (*text*) – Name of the inventory

Return type str

`getFirstUrn`(*urn*, *inventory=None*)
Retrieve the first passage urn of a text

Parameters

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

Return type str

getLabel(*urn*, *inventory=None*)
Retrieve informations about a CTS Urn

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory

Return type str

getPassage(*urn*, *inventory=None*, *context=None*)
Retrieve a passage

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory
- **context** (*int*) – Number of citation units at the same level of the citation hierarchy as the requested urn, immediately preceding and immediately following the requested urn to include in the reply

Return type str

getPassagePlus(*urn*, *inventory=None*, *context=None*)
Retrieve a passage and informations about it

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory
- **context** (*int*) – Number of citation units at the same level of the citation hierarchy as the requested urn, immediately preceding and immediately following the requested urn to include in the reply

Return type str

getPrevNextUrn(*urn*, *inventory=None*)
Retrieve the previous and next passage urn of one passage

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory

Return type str

getValidReff(*urn*, *inventory=None*, *level=None*)
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 XML Response from the API as string

Return type str

Prototypes

class `MyCapytain.retrievers.proto.`**`API`**(*endpoint*)

Bases: `object`

API Prototype object

Parameters

- **`self`** (`API`) – Object
- **`endpoint`** (`text`) – URL of the API

Variables **`endpoint`** – Url of the endpoint

class `MyCapytain.retrievers.proto.`**`Ahab`**(*endpoint*)

Bases: *`MyCapytain.retrievers.proto.API`*

Abstract Capitains Ahab API See : http://capitains.github.io/pages/ahab.html

`permalink`(*urn*, *format='xml'*)

Perform a permalink request on API

Return type str

`search`(*query*, *urn*, *start=1*, *limit=5*, *format='json'*)

Perform a search on given namespace

Parameters

- **`query`** (`text`) – Term to perform search on
- **`urn`** (`text`) – Partial or complete urn identifying the request
- **`start`** (*`int`*) – Starting element to display
- **`limit`** (*`int`*) – Limit of result displayed
- **`format`** (*`str`*) – Format to request (json or xml)

Return type str

class `MyCapytain.retrievers.proto.`**`CTS`**(*endpoint*)

Bases: *`MyCapytain.retrievers.proto.API`*

CTS API Endpoint Prototype

`getCapabilities`(*inventory*)

Retrieve the inventory information of an API

Parameters **`inventory`** (`text`) – Name of the inventory

Return type str

`getFirstUrn`(*urn*, *inventory*)

Retrieve the first passage urn of a text

Parameters

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

Return type str

getLabel(*urn*, *inventory*)
Retrieve informations about a CTS Urn

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory

Return type str

getPassage(*urn*, *inventory*, *context=None*)
Retrieve a passage

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory
- **context** (*int*) – Number of citation units at the same level of the citation hierarchy as the requested urn, immediately preceding and immediately following the requested urn to include in the reply

Return type str

getPassagePlus(*urn*, *inventory*, *context=None*)
Retrieve a passage and informations about it

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory
- **context** (*int*) – Number of citation units at the same level of the citation hierarchy as the requested urn, immediately preceding and immediately following the requested urn to include in the reply

Return type str

getPrevNextUrn(*urn*, *inventory*)
Retrieve the previous and next passage urn of one passage

Parameters

- **urn** (text) – URN identifying the text's passage (Minimum depth : 1)
- **inventory** (text) – Name of the inventory

Return type str

getValidReff(*urn*, *inventory*, *level=1*)
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

Return type str

2.5.3 Texts and inventories

Text

TEI based texts

class `MyCapytain.resources.texts.tei.`**`Citation`**(*name=None*, *xpath=None*, *scope=None*, *refsDecl=None*, *child=None*)

Bases: *`MyCapytain.common.reference.Citation`*

Implementation of Citation for TEI markup

static **`ingest`**(*resource*, *xpath='.//tei:cRefPattern'*)

Ingest a resource and store data in its instance

Parameters

- **`resource`** (*`lxml.etree._Element`*) – XML node cRefPattern or list of them in ASC hierarchy order (deepest to highest, eg. lines to poem to book)
- **`xpath`** (*`str`*) – XPath to use to retrieve citation

Returns A citation object

Return type *Citation*

class `MyCapytain.resources.texts.tei.`**`Passage`**(*parent=None*, ***kwargs*)

Bases: *`MyCapytain.resources.proto.text.Passage`*

`text`(*exclude=None*)

Text content of the passage

Parameters **`exclude`** (*`List`*) – Remove some nodes from text

Return type basestring

Returns Text of the xml node

Example

```
>>>    P = Passage(resource='<l n="8">Ibis <note>hello<a>b</a></note> ab excusso miss
>>>    P.text == "Ibis hello b ab excusso missus in astra sago. "
>>>    P.text(exclude=["note"]) == "Ibis hello b ab excusso missus in astra sago. "
```

`xml`

XML Representation of the Passage

Return type lxml.etree._Element

Returns XML element representing the passage

Locally read text

class `MyCapytain.resources.texts.local.`**`Text`**(*urn=None*, *citation=None*, *resource=None*, *autoreffs=False*)

Bases: *`MyCapytain.resources.proto.text.Text`*

Implementation of CTS tools for local files

Parameters

- **`urn`** (`MyCapytain.common.reference.URN`) – A URN identifier

- **resource** (*lxml.etree._Element*) – A resource
- **citation** (MyCapytain.common.reference.Citation) – Highest Citation level
- **autoreffs** (*bool*) – Parse references on load (default : True)

Variables **resource** – lxml

citation
Get the lowest cRefPattern in the hierarchy

Return type *Citation*

getPassage(*reference*, *hypercontext=True*)
Finds a passage in the current text

Parameters

- **reference** (*list, Reference*) – Identifier of the subreference / passages
- **hypercontext** (*bool*) – If set to true, retrieves nodes up to the given one, cleaning non required siblings.

Return type Passage, ContextPassage

Returns Asked passage

Note: As of MyCapytain 0.1.0, Text().getPassage() returns by default a ContextPassage, thus being able to handle range. This design change also means that the returned tree is way different that a classic

Passage. To retrieve MyCapytain<=0.0.9 behaviour, use *hypercontext=False*.

getValidReff(*level=None*, *reference=None*, *_debug=False*)
Retrieve valid passages directly

Parameters

- **level** (*int*) – Depth required. If not set, should retrieve first encountered level (1 based)
- **reference** (Reference) – Passage Reference
- **_debug** (*bool*) – Check on passages duplicates

Returns List of levels

Return type list(basestring, str)

Note: GetValidReff works for now as a loop using Passage, subinstances of Text, to retrieve the valid

informations. Maybe something is more powerfull ?

nested_dict(*exclude=None*)
Nested Dict Representation of the text passages

Parameters **exclude** (*List*) – Remove some nodes from text according to *MyCapytain.resources.texts.tei.Passage.text*

Return type dict

Returns Dictionary

parse()
  Parse the object and generate the children

text(*exclude=None*)
  Returns the text of the XML resource without the excluded nodes

  Parameters **exclude** (*list (str)*) – List of nodes

  Returns Text of the text without the text inside removed nodes

  Return type str

class MyCapytain.resources.texts.local.**Passage**(*urn=None*, *resource=None*, *parent=None*, *citation=None*, *reference=None*)
  Bases: *MyCapytain.resources.texts.tei.Passage*

  Passage class for local texts which is fast but contains the minimum DOM.

  For design purposes, some people would prefer passage to be found quickly (Text indexing for example). Passage keeps only the node found through the xpath

  Example : for a text with a citation scheme with following refsDecl : */TEI/text/body/div[@type='edition']/div[@n='$1']/div[@n='$2']/l[@n='$3']* and a passage 1.1.1, this class will build an XML tree looking like the following

```
<l n='1'>Lorem ipsum</l>
```

  Parameters

  - **urn** (URN) – A URN identifier
  - **resource** (*etree._Element*) – A resource
  - **parent** (Passage) – Parent of the current passage
  - **citation** (Citation) – Citation for children level
  - **reference** (*Reference, List*) – Identifier of the subreference without URN information

  Warning: This passage system does not accept range

children
  Children of the passage

  Returns Dictionary of chidren, where key are subreferences

  Return type OrderedDict

first
  First child of current Passage

  Returns None if current Passage has no children, first child passage if available

  Return type None, Passage

get(*key=None*)
  Get a child or multiple children

  Parameters **key** (*basestring or int*) – String identifying a passage

  Raises **KeyError** – When key identifies a child unknown to this passage

  Return type List.Passage

Returns List of passage identified by key. If key is None, returns all children

Note: Call time depends on parsing status. If the passage was never parsed, then on first call citation is used to find children

last
Last child of current Passage

Returns None if current Passage has no children, last child passage if available

Return type None, Passage

next
Next passage

Returns Next passage at same level

Return type *Passage*

prev
Previous passage

Returns Previous passage at same level

Return type *Passage*

reference
Id represents the passage subreference as a list of basestring

Returns Representation of the passage subreference as a list

Return type *Reference*

urn
URN Identifier of the object

Return type *URN*

class `MyCapytain.resources.texts.local.`**`ContextPassage`**(*urn=None*, *resource=None*, *parent=None*, *citation=None*, *reference=None*)

Bases: *`MyCapytain.resources.texts.local.Passage`*

Passage class for local texts which rebuilds the tree up to the passage.

For design purposes, some people would prefer the output of GetPassage to be consistent. ContextPassage rebuilds the tree of the text up to the passage, keeping attributes of original nodes

Example : for a text with a citation scheme with following refsDecl : */TEI/text/body/div[@type='edition']/div[@n='$1']/div[@n='$2']/l[@n='$3']* and a passage 1.1.1-1.2.3, this class will build an XML tree looking like the following

```
<TEI ...>
    <text ...>
        <body ...>
            <div type='edition' ...>
                <div n='1' ...>

                    <div n='1' ...>
                        <l n='1'>...</l>
                        ...
                    </div>
                    <div n='2' ...>
```

```
                              <l n='3'>...</l>
                          </div>
                      </div>
                  </div>
              </body>
          </text>
      </TEI>
```

Parameters

- **`urn`** (`URN`) – URN of the source text or of the passage
- **`resource`** (*`etree._Element, Text`*) – Element representing the passage
- **`parent`** (`Text`) – Text containing the passage
- **`citation`** (`Citation`) – Citation scheme of the text
- **`reference`** (`Reference`) – Passage reference

Note: .prev, .next, .first and .last won't run on passage with a range made of two different level, such as 1.1-1.2.3 or 1-a.b. Those will raise *InvalidSiblingRequest*

`children`
Children of the passage

Return type None, Reference

Returns Dictionary of chidren, where key are subreferences

`first`
First child of current Passage

Returns None if current Passage has no children, first child passage if available

Return type None, Reference

`last`
Last child of current Pass

Returns None if current Passage has no children, last child passage if available

Return type None, Reference

`next`
Next passage

Returns Next passage at same level

Return type None, Reference

`prev`
Get the Previous passage reference

Returns Previous passage reference at the same level

Return type None, Reference

`text`(*exclude=None*)
Text content of the passage

Parameters **`exclude`** (*`List`*) – Remove some nodes from text

Return type basestring

Returns Text of the xml node

Example

```
>>>    P = Passage(resource='<l n="8">Ibis <note>hello<a>b</a></note> ab excusso miss
>>>    P.text == "Ibis hello b ab excusso missus in astra sago. "
>>>    P.text(exclude=["note"]) == "Ibis ab excusso missus in astra sago. "
```

`tostring`(**args*, ***kwargs*)

Transform the Passage in XML string

Parameters

- **`args`** – Ordered arguments for etree.tostring() (except the first one)
- **`kwargs`** – Named arguments

Returns

`xpath`(**args*, ***kwargs*)

Perform XPath on the passage XML

Parameters

- **`args`** – Ordered arguments for etree._Element().xpath()
- **`kwargs`** – Named arguments

Returns Result list

Return type list(etree._Element)

API's Text results

class `MyCapytain.resources.texts.api.`**`Text`**(*urn*, *resource*, *citation=None*, ***kwargs*)

Bases: *`MyCapytain.resources.proto.text.Text`*

Passage representing object prototype

Parameters

- **`urn`** (`MyCapytain.common.reference.URN`) – A URN identifier
- **`resource`** (`MyCapytain.retrievers.proto.CTS`) – An API endpoint
- **`citation`** (`MyCapytain.resources.texts.tei.Citation`) – Citation for children level
- **`id`** (*`List`*) – Identifier of the subreference without URN informations

`DEFAULT_LANG` **= u'eng'**

`getFirstUrn`(*reference=None*)

Get the first children URN for a given resource

Parameters **`reference`** (*`Reference, str`*) – Reference from which to find child (If None, find first reference)

Returns Children URN

Return type *URN*

getLabel()
Retrieve metadata about the text

Return type *Metadata*

Returns Dictionary with label informations

getPassage(*reference=None*)
Retrieve a passage and store it in the object

Parameters **reference** (*MyCapytain.common.reference.Reference, or MyCapytain.common.reference.URN, or str or list(str)*) – Reference of the passage

Return type *Passage*

Returns Object representing the passage

Raises *TypeError* when reference is not a list or a Reference

getPassagePlus(*reference=None*)
Retrieve a passage and informations around it and store it in the object

Parameters **reference** (*MyCapytain.common.reference.Reference or List of basestring*) – Reference of the passage

Return type *Passage*

Returns Object representing the passage

Raises *TypeError* when reference is not a list or a Reference

getPrevNextUrn(*reference*)
Get the previous URN of a reference of the text

Parameters **reference** (`Reference`) – Reference from which to find siblings

Returns (Previous Passage Reference,Next Passage Reference)

getValidReff(*level=1*, *reference=None*)
Given a resource, Text will compute valid reffs

Parameters

- **level** (*Int*) – Depth required. If not set, should retrieve first encountered level (1 based)
- **reference** (`Reference`) – Passage reference

Return type list(str)

Returns List of levels

reffs
Get all valid reffs for every part of the Text

Return type *MyCapytain.resources.texts.tei.Citation*

class MyCapytain.resources.texts.api.**Passage**(*urn*, *resource*, **args*, ***kwargs*)
Bases: *MyCapytain.resources.texts.tei.Passage*

first
Children passage

Return type *Passage*

Returns Previous passage at same level

static **`firstUrn`**(*resource*)

Parse a resource to get the first URN

Parameters **`resource`** (`etree._Element`) – XML Resource

Returns Tuple representing previous and next urn

Return type (URN, URN)

`getFirst`()

Shortcut for getting the first child passage

Return type *Passage*

Returns Previous passage at same level

`getNext`()

Shortcut for getting the following passage

Return type *Passage*

Returns Following passage at same level

`getPrev`()

Shortcut for getting the preceding passage

Return type *Passage*

Returns Previous passage at same level

`next`

Shortcut for getting the following passage

Return type *MyCapytain.common.reference.Reference*

Returns Following passage reference

`prev`

Previous passage

Return type *Passage*

Returns Previous passage at same level

static **`prevnext`**(*resource*)

Parse a resource to get the prev and next urn

Parameters **`resource`** (`etree._Element`) – XML Resource

Returns Tuple representing previous and next urn

Return type (URN, URN)

Inventories

class `MyCapytain.resources.inventory.`**`Citation`**(*name=None*, *xpath=None*, *scope=None*, *refsDecl=None*, *child=None*)

Bases: `MyCapytain.common.reference.Citation`

Citation XML implementation for TextInventory

`escape` **= <_sre.SRE_Pattern object>**

static **`ingest`**(*resource*, *element=None*, *xpath=u'ti:citation'*)

Ingest xml to create a citation

Parameters

- **`xml`** – XML on which to do xpath
- **`element`** – Element where the citation should be stored
- **`xpath`** – XPath to use to retrieve citation

Returns Citation

`MyCapytain.resources.inventory.`**`Edition`**(*resource=None*, *urn=None*, *parents=None*)
Create an edition subtyped Text object

class `MyCapytain.resources.inventory.`**`Text`**(***kwargs*)
Bases: *`MyCapytain.resources.proto.inventory.Text`*

Represents a CTS Text

..automethod:: __str__

`export`(*output=u'xml'*, ***kwargs*)
Create a {format} version of the Work

Parameters **`output`** (*`basestring, citation`*) – Format to be chosen (Only XML for now)

Return type lxml.etree._Element

Returns XML representation of the object

`parse`(*resource*)
Parse a resource to feed the object

Parameters **`resource`** (*`basestring or lxml.etree._Element`*) – An xml representation object

Returns None

class `MyCapytain.resources.inventory.`**`TextGroup`**(***kwargs*)
Bases: *`MyCapytain.resources.proto.inventory.TextGroup`*

Represents a CTS Textgroup in XML

`export`(*output=u'xml'*)
Create a {format} version of the TextInventory

Parameters **`output`** (*`basestring`*) – Format to be chosen (Only XML for now)

Return type lxml.etree._Element

Returns XML representation of the object

`parse`(*resource*)
Parse a resource

Parameters

- **`resource`** – Element rerpresenting the textgroup
- **`type`** – basestring, etree._Element

class `MyCapytain.resources.inventory.`**`TextInventory`**(***kwargs*)
Bases: *`MyCapytain.resources.proto.inventory.TextInventory`*

Represents a CTS Inventory file

`export`(*output=u'xml'*)
Create a {output} version of the TextInventory

Parameters **output** (*basestring*) – output to be chosen (Only XML for now)

Return type lxml.etree._Element

Returns XML representation of the object

parse (*resource*)
Parse a resource

Parameters

- **resource** – Element representing the text inventory
- **type** – basestring, etree._Element

MyCapytain.resources.inventory.**Translation** (*resource=None*, *urn=None*, *parents=None*)
Create a translation subtyped Text object

class MyCapytain.resources.inventory.**Work** (***kwargs*)
Bases: *MyCapytain.resources.proto.inventory.Work*

Represents a CTS Textgroup in XML

..automethod:: __str__

export (*output=u'xml'*)
Create a {format} version of the Work

Parameters **output** (*basestring*) – Format to be chosen (Only XML for now)

Return type lxml.etree._Element

Returns XML representation of the object

parse (*resource*)
Parse a resource

Parameters

- **resource** – Element rerpresenting a work
- **type** – basestring, etree._Element

MyCapytain.resources.inventory.**xpathDict** (*xml*, *xpath*, *children*, *parents*, ***kwargs*)
Returns a default Dict given certain informations

Parameters

- **xml** (*etree*) – An xml tree
- **xpath** – XPath to find children
- **children** (inventory.Resource) – Object identifying children
- **parents** (*tuple.<inventory.Resource>*) – Tuple of parents

Return type collections.defaultdict.<basestring, inventory.Resource>

Returns Dictionary of children

Prototypes

class MyCapytain.resources.proto.text.**Passage** (*parent=None*, ***kwargs*)
Bases: *MyCapytain.resources.proto.text.Resource*

Passage representing object prototype

Parameters

- **urn** (`MyCapytain.common.reference.URN`) – A URN identifier
- **resource** (*lxml.etree._Element*) – A resource
- **parent** (`MyCapytain.resources.texts.tei.Passage`) – Parent of the current passage
- **citation** (`MyCapytain.resources.texts.tei.Citation`) – Citation for children level
- **id** (*List*) – Identifier of the subreference without URN informations

children
Children of the passage

Return type OrderedDict

Returns Dictionary of chidren, where key are subreferences

first
First child of current Passage

Return type None or Passage

Returns None if current Passage has no children, first child passage if available

last
Last child of current Passage

Return type None or Passage

Returns None if current Passage has no children, last child passage if available

next
Following passage

Return type *Passage*

Returns Following passage at same level

prev
Previous passage

Return type *Passage*

Returns Previous passage at same level

class `MyCapytain.resources.proto.text.`**PassagePlus**(*passage*, *prev*, *next*)
Bases: `tuple`

next
Alias for field number 2

passage
Alias for field number 0

prev
Alias for field number 1

class `MyCapytain.resources.proto.text.`**Resource**(*urn=None*, *resource=None*)
Bases: `object`

Initiate a Resource object

Parameters

- **urn** (MyCapytain.common.reference.URN) – A URN identifier
- **resource** (*Any*) – A resource

urn
URN Identifier of the object

Return type *MyCapytain.common.reference.URN*

class MyCapytain.resources.proto.text.**Text** (*citation=None*, *metadata=None*, ***kwargs*)
Bases: *MyCapytain.resources.proto.text.Resource*

A CTS Text

citation
Get the lowest cRefPattern in the hierarchy

Return type *MyCapytain.common.reference.Citation*

getLabel ()
Retrieve metadata about the text

Return type dict

Returns Dictionary with label informations

getPassage (*reference*)
Retrieve a passage and store it in the object

Parameters **reference** (*MyCapytain.common.reference.Reference or List of basestring*) – Reference of the passage

Return type *Passage*

Returns Object representing the passage

Raises *TypeError* when reference is not a list or a Reference

getValidReff (*level=1*, *reference=None*)
Given a resource, Text will compute valid reffs

Parameters

- **level** (*Int*) – Depth required. If not set, should retrieve first encountered level (1 based)
- **passage** (Reference) – Subreference (optional)

Return type List.basestring

Returns List of levels

reffs
Get all valid reffs for every part of the Text

Return type *MyCapytain.resources.texts.tei.Citation*

MyCapytain.resources.proto.inventory.**Edition** (*resource=None*, *urn=None*, *parents=None*)

class MyCapytain.resources.proto.inventory.**Resource** (*resource=None*)
Bases: object

Resource represents any resource from the inventory

export (*format=None*)

parse (*resource*)
Parse the object resource

Parameters **`resource`** (*`Any`*) – Resource representing the TextInventory

Return type List

`setResource`(*resource*)
Set the object property resource

Parameters **`resource`** (*`Any`*) – Resource representing the TextInventory

Return type Any

Returns Input resource

class `MyCapytain.resources.proto.inventory.`**`Text`**(*resource=None*, *urn=None*, *parents=None*, *subtype='Edition'*)
Bases: *`MyCapytain.resources.proto.inventory.Resource`*

Represents a CTS Text

`editions`()
Get all editions of the texts

Returns List of editions

Return type [Text]

`translations`(*key=None*)
Get translations in given language

Parameters **`key`** – Language ISO Code to filter on

Returns

class `MyCapytain.resources.proto.inventory.`**`TextGroup`**(*resource=None*, *urn=None*, *parents=None*)
Bases: *`MyCapytain.resources.proto.inventory.Resource`*

Represents a CTS Textgroup

class `MyCapytain.resources.proto.inventory.`**`TextInventory`**(*resource=None*, *id=None*)
Bases: *`MyCapytain.resources.proto.inventory.Resource`*

Represents a CTS Inventory file

`MyCapytain.resources.proto.inventory.`**`Translation`**(*resource=None*, *urn=None*, *parents=None*)

class `MyCapytain.resources.proto.inventory.`**`Work`**(*resource=None*, *urn=None*, *parents=None*)
Bases: *`MyCapytain.resources.proto.inventory.Resource`*

Represents a CTS Work

`getLang`(*key=None*)
Find a translation with given language

Parameters **`key`** (*`basestring`*) – Language to find

Return type [Text]

Returns List of availables translations

2.6 Benchmarks

In the recent attempt to boost our system, we had a look on the performance of MyCapytain with different parser. Even if as 1.0.1 xmlparser() is the recommended tool, we highly recommend to switch to lxml.objectify.parse() parser for performance. In the following benchmark run with timeit.sh on the main repo (You need PerseusDL/canonical-latinLit somewhere), the first line is run with lxml.etree, the second with objectify and the third with a pickled object.

Testing on Seneca, Single Simple Passage

- 100 loops, best of 3: 4.45 msec per loop
- 100 loops, best of 3: 4.15 msec per loop
- 100 loops, best of 3: 3.75 msec per loop

Testing range

- 100 loops, best of 3: 7.63 msec per loop
- 100 loops, best of 3: 7.72 msec per loop
- 100 loops, best of 3: 6.66 msec per loop

Testing with a deeper architecture

- 100 loops, best of 3: 18.2 msec per loop
- 100 loops, best of 3: 14.3 msec per loop
- 100 loops, best of 3: 9.31 msec per loop

Testing with a deeper architecture at the end

- 100 loops, best of 3: 18.2 msec per loop
- 100 loops, best of 3: 14.2 msec per loop
- 100 loops, best of 3: 9.34 msec per loop

Testing with a deeper architecture with range

- 100 loops, best of 3: 19.3 msec per loop
- 100 loops, best of 3: 14.3 msec per loop
- 100 loops, best of 3: 9.9 msec per loop

Testing with complicated XPATH

- 100 loops, best of 3: 751 usec per loop
- 100 loops, best of 3: 770 usec per loop
- 100 loops, best of 3: 617 usec per loop

CHAPTER 3

Indices and tables

- Importing Modules
- genindex
- modindex
- search