MyCapytains Documentation

Release 0.0.1

Thibault Clérice

December 22, 2016

Contents

MyCapytain is a python library which provides a large set of methods to interact with Text Services API such as the Canonical Text Services, the Distributed Text Services. It also provides a programming interface to exploit local textual resources developed according to the Capitains Guidelines.

CHAPTER 1

Simple Example of what it does

The following code and example is badly displayed at the moment on Github. We recommend you to go to http://mycapytain.readthedocs.org

On Leipzig DH Chair's Canonical Text Services API, we can find the Epigrammata of Martial. This texts are identified by the identifier "urn:cts:latinLit:phi1294.phi002.perseus-lat2". We want to have some information about this text so we are gonna ask the API to give its metadata to us :

Listing 1.1: example.py from the Github Repository

```
from MyCapytain.resolvers.cts.api import HttpCTSResolver
from MyCapytain.retrievers.cts5 import CTS
from MyCapytain.common.constants import Mimetypes

# We set up a resolver which communicates with an API available in Leipzig
resolver = HttpCTSResolver(CTS("http://cts.dh.uni-leipzig.de/api/cts/"))
# We require some metadata information
textMetadata = resolver.getMetadata("urn:cts:latinLit:phi1294.phi002.perseus-lat2")
# Texts in CTS Metadata have one interesting property : its citation scheme.
# Citation are embedded objects that carries information about how a text can be quoted, what depth
print(type(textMetadata), [citation.name for citation in textMetadata.citation])
```

This query will return the following information :

```
<class 'MyCapytain.resources.collections.cts.Text'> ['book', 'poem', 'line']
```

```
# Now, we want to retrieve the first line of poem seventy two of the second book
passage = resolver.getTextualNode("urn:cts:latinLit:phi1294.phi002.perseus-lat2", subreference="2.72
# And we want to have its content exported to plain text and have the siblings of this passage (previ
print(passage.export(Mimetypes.PLAINTEXT), passage.siblingsId)
```

And we will get

```
Hesterna factum narratur, Postume, cena
```

If you want to play more with this, like having a list of what can be found in book three, you could go and do

```
poemsInBook3 = resolver.getReffs("urn:cts:latinLit:phi1294.phi002.perseus-lat2", subreference="3")
print(poemsInBook3)
```

Which would be equal to :

```
['3.1', '3.2', '3.3', '3.4', '3.5', '3.6', '3.7', '3.8', '3.9', '3.10', '3.11', '3.12', '3.13', ...]
```

Now, it's your time to work with the resource ! See the CapiTainS Classes page on ReadTheDocs to have a general introduction to MyCapytain objects !

CHAPTER 2

Installation and Requirements

The best way to install MyCapytain is to use pip. MyCapytain tries to support Python over 3.4.

The work needed for supporting Python 2.7 is mostly done, however, since 2.0.0, we are giving up on ensuring that MyCapytain will be compatible with Python < 3 while accepting PR which would help doing so.

```
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 3

Contents

3.1 MyCapytain's Main Objects Explained

3.1.1 Exportable Parent Classes

Description

`MyCapytain.common.constants.Exportable`

The Exportable class is visible all across the library. It provides a common, standardized way to retrieve in an API fashion to what can an object be exported and to exports it. Any exportable object should have an EXPORT_TO constant variable and include a __export__(output, **kwargs) methods if it provides an export type.

Example

The following code block is a mere example of how to implement Exportable and what are its responsibilities. Exportabletypically loops over all the parents class of the current class until it find one exportable system matching the required one.

```
from MyCapytain.common.constants import Exportable, Mimetypes


class Sentence(Exportable):
    """ This class represent a Sentence

    :param content: Content of the sentence
    """
    # EXPORT_TO is a list of Mimetype the object is capable to export to
    EXPORT_TO = [
        Mimetypes.PLAINTEXT, Mimetypes.XML.Std
    ]
    DEFAULT_EXPORT = Mimetypes.PLAINTEXT

    def __init__(self, content):
        self.content = content

    def __export__(self, output=None, **kwargs):
        """ Export the collection item in the Mimetype required.

        :param output: Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)
        :type output: str
```

```
        :return: Object using a different representation
        """
        if output == Mimetypes.PLAINTEXT:
            return self.content
        elif output == Mimetypes.XML.Std:
            return "<sentence>{}</sentence>".format(self.content)


class TEISentence(Sentence):
    """ This class represent a Sentence but adds some exportable accepted output

    :param content: Content of the sentence
    """
    EXPORT_TO = [
        Mimetypes.JSON.Std
    ]

    def __export__(self, output=None, **kwargs):
        """ Export the collection item in the Mimetype required.

        :param output: Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)
        :type output: str
        :return: Object using a different representation
        """
        if output == Mimetypes.JSON.Std:
            return {"http://www.tei-c.org/ns/1.0/sentence": self.content}
        elif output == Mimetypes.XML.Std:
            return "<sentence xmlns=\"http://www.tei-c.org/ns/1.0\">{}</sentence>".format(self.conter


s = Sentence("I love Martial's Epigrammatas")
print(s.export(Mimetypes.PLAINTEXT))
# I love Martial's Epigrammatas
print(s.export())  # Defaults to PLAINTEXT
# I love Martial's Epigrammatas
print(s.export(Mimetypes.XML.Std))
# <sentence>I love Martial's Epigrammatas</sentence>

tei = TEISentence("I love Martial's Epigrammatas")
print(tei.export(Mimetypes.PLAINTEXT))
# I love Martial's Epigrammatas
print(tei.export())  # Defaults to PLAINTEXT
# I love Martial's Epigrammatas
print(tei.export(Mimetypes.JSON.Std))
# {"http://www.tei-c.org/ns/1.0/sentence": I love Martial's Epigrammatas}
print(tei.export(Mimetypes.XML.Std))  # Has been rewritten by TEISentence
# <sentence xmlns="http://www.tei-c.org/ns/1.0">I love Martial's Epigrammatas</sentence>
try:
    print(tei.export(Mimetypes.XML.RDF))
except NotImplementedError as error:
    print(error)
# Raise the error and prints "Mimetype application/rdf+xml has not been implemented for this resource
```

3.1.2 Retrievers

MyCapytain.retrievers.prototypes.API

Description

Retrievers are classes that help build requests to API and return standardized responses from them. There is no real perfect prototypes. The only requirements for a Retriever is that its query function should returns string only. It is not the role of the retrievers to parse response. It is merely to facilitate the communication to remote API most of the time.

Recommendations

For Textual API, it is recommended to implement the following requests

- getTextualNode(textId[str], subreference[str], prevnext[bool], metadata[bool])
- getMetadata(objectId[str], **kwargs)
- getSiblings(textId[str], subreference[str])
- getReffs(textId[str], subreference[str], depth[int])

Example of implementation : CTS 5

`MyCapytain.retrievers.cts5.CTS`

```
from MyCapytain.retrievers.cts5 import CTS

# We set up a retriever which communicates with an API available in Leipzig
retriever = CTS("http://cts.dh.uni-leipzig.de/api/cts/")
# We require a passage : passage is now a Passage object
passage = retriever.getPassage("urn:cts:latinLit:phi1294.phi002.perseus-lat2:1.1")
# Passage is now equal to the string content of http://cts.dh.uni-leipzig.de/api/cts/?request=GetPass
print(passage)

"""
<GetPassage><request><requestName>GetPassage</requestName><requestUrn>urn:cts:latinLit:phi1294.phi00
<reply><urn>urn:cts:latinLit:phi1294.phi002.perseus-lat2:1.1</urn><passage><TEI>
<text n="urn:cts:latinLit:phi1294.phi002.perseus-lat2" xml:id="stoa0045.stoa0"><body>
<div type="edition" n="urn:cts:latinLit:phi1294.phi002.perseus-lat2" xml:lang="lat">
<div type="textpart" subtype="book" n="1"><div type="textpart" subtype="poem" n="1">
<head>I</head>
<l n="1">Hic est quem legis ille, quem requiris, </l>
<l n="2">Toto notus in orbe Martialis </l>
<l n="3">Argutis epigrammaton libellis: <pb/></l>
<l n="4">Cui, lector studiose, quod dedisti </l>
<l n="5">Viventi decus atque sentienti, </l>
<l n="6">Rari post cineres habent poetae. </l>
</div></div></div></body></text></TEI></passage></reply>
"""
```

3.1.3 Text and Passages

Description

Hierarchy

The generic idea of both Text and Passage's classes is that they inherit from a longer trail of text bearing object that complexified over different features. The basic is

- *TextualElement* is an object which can bear Metadata and Collection information. It has a .text property and is exportable
- *TextualNode* inherits from NodeId and unlike TextualElement, TextualNode is part of a graph of CitableObject. It bears informations about its siblings, parents, children.
- *TextualGraph* is a bit interactive : you can query for children nodes and get descendant references of the object.
- *InteractiveTextualNode* is completely interative . You can browse the graph by accessing the `.next` property for example : it should then return an InteractiveTextualNode as well
- *CTSNode* has two unique methods more as well as a `urn` property.
- From *CTSNode* we find *CitableText* and *Passage*, which represents complete and portion of a Text. The main difference is that CitableText has no parents, no siblings.

Fig. 3.1: Prototype of Texts from **:module:'MyCapytain.resources.prototypes.text'**. `NodeId` and `Exportable` are respectively from **:module:'MyCapytain.common.reference'** and **:module:'MyCapytain.common.constants'**.

Objectives

Text and Passages object have been built around InteractiveTextualNode which fills the main purpose of MyCapytain :being able to interact with citable, in-graph texts that are retrieve through web API or local files. Any implementation should make sure that the whole set of navigation tool are covered. Those are :

Tree Identifiers(Returns str Identifiers)	Tree Navigations (Returns InteractiveTextualNode or children class)	Retrieval Methods	Other
prevId	prev	.getTextualNode(subreference)	id : TextualNode Identifier [str]
nextId	nextId	.getReffs(subreference[optional])	metadata : Metadata Informations [Metadata]
siblingsId [tuple[str]]	siblings [tuple[InteractiveTextualNode]]		about : Collection Information [Collection]
parentId	parent		citation : Citation Information [Citation]
childIds [list[str]]	children [list[InteractiveTextualNode]]		text : String Representation of the text without annotation
firstId	first		.export()
lastId	last		

The encodings module

The encoding module contains special implementations : they technically do not support interactive methods but provides generic parsing and export methods for specific type of contents such as TEI XML object or other formats such as json, csv, treebank objects in the future.

The `TEIResource` for example requires the object to be set up with a resource parameters that will be furtherparsed using lxml. From there, it provides export such as plain/text, TEI XML, nested dictionaries or even anlxml etree interface.

Implementation example : HTTP API Passage work

```
from MyCapytain.retrievers.cts5 import CTS
from MyCapytain.resources.texts.api.cts import Text

# We set up a retriever which communicates with an API available in Leipzig
retriever = CTS("http://cts.dh.uni-leipzig.de/api/cts/")

# Given that we have other examples that shows how to work with text,
# we will focus here on playing with the graph functionality of texts implementations.
# We are gonna retrieve a text passage and the retrieve all its siblings in different fashion#
# The main point is to find all children of the same parent.
# The use case could be the following : some one want to retrieve the full text around a citation
# To enhance the display a little.

# We will work with the line 7 of poem 39 of book 4 of Martial's Epigrammata
# The text is urn:cts:latinLit:phi1294.phi002.perseus-lat2
text = Text(retriever=retriever, urn="urn:cts:latinLit:phi1294.phi002.perseus-lat2")

# We retrieve up the passage
target = text.getTextualNode(subreference="4.39.7")
print(target.text)
"""
Nec quae Callaico linuntur auro,
"""

# The parent way :
# - get to the parent,
# - retrieve each node,
# - print only the one which are not target

parent = target.parent
for node in parent.children:
    if node.id != target.id:
        print("{}\t{}".format(node.id, node.text))
    else:
        print("------Original Node-----------")

"""
4.39.1        Argenti genus omne comparasti,
4.39.2        Et solus veteres Myronos artes,
4.39.3        Solus Praxitelus manum Scopaeque,
4.39.4        Solus Phidiaci toreuma caeli,
4.39.5        Solus Mentoreos habes labores.
4.39.6        Nec desunt tibi vera Gratiana,
------Original Node-----------
4.39.8        Nec mensis anaglypta de paternis.
4.39.9        Argentum tamen inter omne miror
4.39.10        Quare non habeas, Charine, purum.
"""

print("\n\nSecond Method\n\n")

# We are gonna do another way this time :
# - get the previous until we change parent
# - get the next until we change parent

parentId = node.parentId
```

```
# Deal with the previous ones
current = target.prev
while current.parentId == parentId:
    print("{}\t{}".format(current.id, current.text))
    current = current.prev

print("------Original Node-----------")

# Deal with the next ones
current = target.next
while current.parentId == parentId:
    print("{}\t{}".format(current.id, current.text))
    current = current.next
"""
4.39.6        Nec desunt tibi vera Gratiana,
4.39.5        Solus Mentoreos habes labores.
4.39.4        Solus Phidiaci toreuma caeli,
4.39.3        Solus Praxitelus manum Scopaeque,
4.39.2        Et solus veteres Myronos artes,
4.39.1        Argenti genus omne comparasti,
------Original Node-----------
4.39.8        Nec mensis anaglypta de paternis.
4.39.9        Argentum tamen inter omne miror
4.39.10        Quare non habeas, Charine, purum.

"""
```

Other Example

See MyCapytain.local

3.1.4 Collection

Description

Collections are the metadata containers object in MyCapytain. Unlike other object, they will never contain textual content such as Texts and Passages but will in return help you browse through the catalog of one APIs collection and identify manually or automatically texts that are of relevant interests to you.

The main informations that you should be interested in are :

- Collections are children from Exportable. As of 2.0.0, any collection can be exported to JSON DTS.
- Collections are built on a hierarchy. They have children and descendants
- Collections have identifiers and title (Main name of what the collection represents : if it's an author, it's her name, a title for a book, a volume label for a specific edition, etc.)
- Collections can inform the machine if it represents a readable object : if it is readable, it means that using its identifier, you can query for passages or references on the same API.

Main Properties

- Collection().id : Identifier of the object
- Collection().title : Title of the object

- Collection().readable : If True, means that the Collection().id can be used in GetReffs or GetTextualNode queries
- Collection().members : Direct children of the object
- Collection().descendants : Direct and Indirect children of the objects
- Collection().readableDescendants : Descendants that have .readable as True
- Collection().export() : Export Method
- Collection().metadata : Metadata object that contain flat descriptive localized informations about the object.

Implementation : CTS Collections

Note: For a recap on what Textgroup means or any CTS jargon, go to http://capitains.github.io/pages/vocabulary

CTS Collections are divided in 4 kinds : TextInventory, TextGroup, Work, Text. Their specificity is that the hierarchyof these objects are predefined and always follow the same order. They implement a special export (`MyCapytain.common.constants.Mimetypes.XML.CTS`) which basically exports to the XML Text Inventory Formatthat one would find making a GetCapabilities request.

CapiTainS CTS Collections implement a parents property which represent a list of parents where .parents' order is equalto `Text.parents = [Work(), TextGroup(), TextInventory()]`).

Their finale implementation accepts to parse resources through the `resource=` named argument.

Example

```
from MyCapytain.retrievers.cts5 import CTS
from MyCapytain.resources.collections.cts import TextInventory, Work
from MyCapytain.common.constants import Mimetypes
from pprint import pprint

"""
In order to have a real life example,
we are gonna query for data in the Leipzig CTS API
We are gonna query for metadata about Seneca who
is represented by urn:cts:latinLit:stoa0255

To retrieve data, we are gonna make a GetMetadata query
to the CTS Retriever.
"""
retriever = CTS("http://cts.dh.uni-leipzig.de/api/cts/")
# We store the response (Pure XML String)
response = retriever.getMetadata(objectId="urn:cts:latinLit:stoa0255")

"""
From here, we actually have the necessary data, we can now
play with collections. TextInventory is the main collection type that is needed to
parse the whole response.
"""
inventory = TextInventory(resource=response)
# What we are gonna do is print the title of each descendant :
for descendant in inventory.descendants:
    # Metadatum resolve any non-existing language ("eng", "lat") to a default one
    # Putting default is just making that clear
    print(descendant.title["default"])
```

```
30
31  """
32  You should see in there things such as
33  -   "Seneca, Lucius Annaeus" (The TextGroup or main object)
34  -   "de Ira" (The Work object)
35  -   "de Ira, Moral essays Vol 2" (The Edition specific Title)
36
37  We can now see other functions, such as the export to JSON DTS.
38  Collections have a unique feature built in : they allow for
39  accessing an item using its key as if it were a dictionary :
40  The identifier of a De Ira is urn:cts:latinLit:stoa0255.stoa0110
41  """
42  deIra = inventory["urn:cts:latinLit:stoa0255.stoa010"]
43  assert isinstance(deIra, Work)
44  pprint(deIra.export(output=Mimetypes.JSON.DTS.Std))
45  # you should see a DTS representation of the work
46
47  """
48  What we might want to do is to browse metadata about seneca's De Ira
49  Remember that CTSCollections have a parents attribute !
50  """
51  for descAsc in deIra.descendants + [deIra] + deIra.parents:
52      # We filter out Textgroup which has an empty Metadata value
53      if not isinstance(descAsc, TextInventory):
54          print(
55              descAsc.metadata.export(output=Mimetypes.JSON.Std)
56          )
57  """
58  And of course, we can simply export deIra to CTS XML format
59  """
60  print(deIra.export(Mimetypes.XML.CTS))
```

3.1.5 Resolvers

Description

Resolvers were introduced in 2.0.0b0 and came as a solution to build tools around Text Services APIs where you can seamlessly switch a resolver for another and not changing your code, join together multiple resolvers, etc. The principle behind resolver is to provide native python object based on API-Like methods which are restricted to four simple commands :

- getTextualNode(textId[str], subreference[str], prevnext[bool], metadata[bool]) -> Passage
- getMetadata(objectId[str], **kwargs) -> Collection
- getSiblings(textId[str], subreference[str]) -> tuple([str, str])
- getReffs(textId[str], subreference[str], depth[int]) -> list([str])

These function will always return objects derived from the major classes, *i.e.* Passage and Collection for the two firsts and simple collections of strings for the two others. Resolvers fills the hole between these base objects and the original retriever objects that were designed to return plain strings from remote or local APIs.

The base functions are represented in the prototype, and only getMetadata might be expanded in terms of arguments depending on what filtering can be offered. Though, any additional filter has not necessarily effects with other resolvers.

Historical Perspective

The original incentive to build resolvers was the situation with retrievers, in the context of the Nautilus API and Nemo UI : Nemo took a retriever as object, which means that, based on the prototype, Nemo was retrieving string objects. That made sense as long as Nemo was running with HTTP remote API because it was actually receiving string objects which were not even (pre-)processed by the Retriever object. But since Nautilus was developed (a fully native python CTS API), we had the situation where Nemo was parsing strings that were exported from python etree objects by Nautilus which parsed strings. Introducing Resolvers, we managed to avoid this double parsing effect in any situation : MyCapytain now provides a default class to provide access to querying text no matter what kind of transactions there is behind the Python object. At the same time, Resolvers provide a now unified system to retrieve texts independently from the retrieverstandard type (CTS, DTS, Proprietary, etc.).

Prototype

class MyCapytain.resolvers.prototypes.**Resolver**

Resolver provide a native python API which returns python objects.

Initiation of resolvers are dependent on the implementation of the prototype

getMetadata(*objectId=None*, ***filters*)

Request metadata about a text or a collection

Parameters

- **objectId** (*str*) – Object Identifier to filter on
- **filters** (*dict*) – Kwargs parameters.

Returns Collection

getReffs(*textId*, *level=1*, *subreference=None*)

Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **level** (*int*) – Depth for retrieval
- **subreference** (*str*) – Passage Reference

Returns List of references

Return type [str]

getSiblings(*textId*, *subreference*)

Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **subreference** (*str*) – Passage Reference

Returns Tuple of references

Return type (str, str)

getTextualNode(*textId*, *subreference=None*, *prevnext=False*, *metadata=False*)

Retrieve a text node from the API

Parameters

- **textId** (*str*) – Text Identifier

- **subreference** (*str*) – Passage Reference
- **prevnext** (*boolean*) – Retrieve graph representing previous and next passage
- **metadata** (*boolean*) – Retrieve metadata about the passage and the text

Returns Passage

Return type *Passage*

Example

```
from MyCapytain.resolvers.cts.api import HttpCTSResolver
from MyCapytain.retrievers.cts5 import CTS
from MyCapytain.common.constants import Mimetypes, NS

# We set up a resolver which communicates with an API available in Leipzig
resolver = HttpCTSResolver(CTS("http://cts.dh.uni-leipzig.de/api/cts/"))
# We require a passage : passage is now a Passage object
# This is an entry from the Smith Myth Dictionary
# The inner methods will resolve to the URI http://cts.dh.uni-leipzig.de/api/cts/?request=GetPassage
# And parse it into interactive objects
passage = resolver.getTextualNode("urn:cts:pdlrefwk:viaf88890045.003.perseus-eng1", "A.abaeus_1")
# We need an export as plaintext
print(passage.export(
    output=Mimetypes.PLAINTEXT
))
"""
    Abaeus ( βαο ), a surname of Apollo
     derived from the town of Abae in Phocis, where the god had a rich temple. (Hesych. s. v.
     βαι ; Hdt. 8.33 ; Paus. 10.35.1 , &c.) [ L.S ]
"""
# We want to find bibliographic information in the passage of this dictionary
# We need an export as LXML ETREE object to perform XPath
print(
    passage.export(
        output=Mimetypes.PYTHON.ETREE
    ).xpath(".//tei:bibl/text()", namespaces=NS, magic_string=False)
)
["Hdt. 8.33", "Paus. 10.35.1"]
```

3.2 Project using MyCapytain

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

3.2.1 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

CLTK Corpora Converter

Capitains Corpora Converter Converts CapiTainS-based Repository (http://capitains.github.io) to JSON for CLTK

3.3 Working with Local CapiTainS XML File

3.3.1 Introduction

The class *MyCapytain.resources.texts.locals.tei.Text* requires the guidelines of Capitains to be implemented in your file.

3.3.2 Example

```
#  We import the correct classes from the local module
from MyCapytain.resources.texts.locals.tei import Text
from MyCapytain.common.constants import Mimetypes, NS
from lxml.etree import tostring

#  We open a file
with open("./tests/testing_data/examples/text.martial.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.getReffs(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
    # We use the _simple parameter to get a fairly simple object
    # Simple makes a straight object that has only the targeted node inside of it
    psg = text.getTextualNode(subreference=ref, simple=True)
    # We print the passage from which we retrieve <note> nodes
    print("\t".join([ref, psg.export(Mimetypes.PLAINTEXT, exclude=["tei:note"])]))

"""
You'll print something like the following :

    1.pr.1        Spero me secutum in libellis meis tale temperamen-
    1.pr.2        tum, ut de illis queri non possit quisquis de se bene
    1.pr.3        senserit, cum salva infimarum quoque personarum re-
    1.pr.4        verentia ludant; quae adeo antiquis auctoribus defuit, ut
```

```
    1.pr.5        nominibus non tantum veris abusi sint, sed et magnis.
    1.pr.6        Mihi fama vilius constet et probetur in me novissimum

"""

# It is possible that what you're interested in is a little more complex
# Like for example, getting a specific text sample with a specific reference
# In TEI !

#  We open another such as Cicero's texts !
with open("./tests/testing_data/examples/text.cicero.xml") as f:
    # We initiate a Text object giving the IO instance to resource argument
    text = Text(resource=f)
    # We are specifically interest in the portion 28-30
    # Note that we won't use 28-30 as cross passage reference won't work properly
    p28_29 = text.getTextualNode("28-29")

    # And we want to be able to work with the xml
    # To be injected in a third party API for lemmatization purposes
    xml = p28_29.export(Mimetypes.XML.Std)
    print("XML of 28-29")
    print(xml)
    print("------------")

    # But what we really want to do, is suppress the note from the XML.
    # So we export to an LXML Object
    document = p28_29.export(Mimetypes.PYTHON.ETREE)
    # We remove some XML
    for element in document.xpath("//tei:note", namespaces=NS):
        element.getparent().remove(element)
    # And we print using LXML constants
    print("Clean XML of 28-29")
    print(tostring(document, encoding=str))
    print("------------")

```

3.4 Known issues and recommendations

3.4.1 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]*

3.5 MyCapytain API Documentation

3.5.1 Utilities, metadata and references

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

Constants

class MyCapytain.common.constants.**Exportable**
Objects that supports Export

Variables ***EXPORT_TO*** – List of Mimetypes the resource can export to

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

class MyCapytain.common.constants.**Mimetypes**
Mimetypes constants that are used to provide export functionality to base MyCapytain object.

Variables

- **JSON** – JSON Resource mimetype
- **XML** – XML Resource mimetype
- **PYTHON** – Python Native Object
- **PLAINTEXT** – Plain string format

class JSON
Json Mimetype

Variables

- **Std** – Standard JSON Export
- **CTS** – CTS Json Export

class DTS
JSON DTS Expression

Variables

- **Std** – Standard DTS Json-LD Expression
- **NoParents** – DTS Json-LD Expression without parents expression

class Mimetypes.**PYTHON**
Python Native Objects

Variables

- **NestedDict** – Nested Dictionary Object
- **ETREE** – Python LXML Etree Object

class MyCapytain
MyCapytain Objects

Variables **ReadableText** – MyCapytain.resources.prototypes.text.CitableText

class Mimetypes.**XML**
XML Mimetype

Variables

- **Std** – Standard XML Export

- **RDF** – RDF XML Expression Export
- **CTS** – CTS API XML Expression Export

class MyCapytain.common.constants.**NAMESPACES**
Namespaces Constants used to provide Namespace capacities across the library

Variables

- **CTS** – CTS Namespace
- **TEI** – TEI Namespace
- **DC** – DC Elements

MyCapytain.common.constants.**NS = {'ahab': 'http://localhost.local', 'ti': 'http://chs.harvard.edu/xmlns/cts', 'xml': 'htt**
List of XPath Namespaces used in guidelines

class MyCapytain.common.constants.**Namespace**(*uri*, *prefix*)
Namespace tuple that can be used to express namespace information

prefix
Alias for field number 1

uri
Alias for field number 0

MyCapytain.common.constants.**RDF_MAPPING = {'http://www.w3.org/2002/07/owl#': 'owl', 'http://www.geonames.org/**
List of RDF URI with their equivalent Prefix

MyCapytain.common.constants.**RDF_PREFIX = {'dbp': 'http://dbpedia.org/property/', 'dbo': 'http://dbpedia.org/ontol**
List of RDF Prefixes with their equivalents

URN, References and Citations

class MyCapytain.common.reference.**NodeId**(*identifier=None*, *children=None*, *parent=None*, *siblings=(None*, *None)*, *depth=None*)
Collection of directional references for a Tree

Parameters

- **identifier** (*str*) – Current object identifier
- **children** (*[str]*) – Current node Children's Identifier
- **parent** (*str*) – Parent of the current node
- **siblings** (*str*) – Previous and next node of the current node
- **depth** (*int*) – Depth of the node in the global hierarchy of the text tree

childIds
Children Node

Return type [str]

depth
Depth of the node in the global hierarchy of the text tree

Return type int

firstId
First child Node

Return type str

id
Current object identifier

Return type str

lastId
Last child Node

Return type str

nextId
Next Node (Sibling)

Return type str

parentId
Parent Node

Return type str

prevId
Previous Node (Sibling)

Return type str

siblingsId
Siblings Node

Return type (str, str)

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

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

Variables

- **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
```

static model()
Generate a standard dictionary model for URN inside function

Returns Dictionary of CTS elements

namespace
CTS Namespace element of the URN

Return type str

Returns Namespace part of the URN

reference
Reference element of the URN

Return type *Reference*

Returns Reference part of the URN

textgroup
Textgroup element of the URN

Return type str

Returns Textgroup part of the URN

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"
```

urn_namespace
General Namespace element of the URN

Return type str

Returns Namespace part of the URN

version
Version element of the URN

Return type str

Returns Version part of the URN

work
Work element of the URN

Return type str

Returns Work part of the URN

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

A reference object giving information

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

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

static **convert_subreference**(*word*, *counter*)

Convert a word and a counter into a standard tuple representation

Parameters

- **word** – Word Element of the subreference
- **counter** – Index of the Word

Returns Tuple representing the element

Return type (str, int)

end

Quick access property for reference end list

Return type *Reference*

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

Return type *Reference*

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*)

A metadatum aggregation object provided to centralize metadata

param keys A metadata field names list

type keys [text_type]

ivar metadata Dictionary of metadatum

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

Parameters **key** (*text_type, 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("title", [("lat", "Amores"), ("fre", "Les Amours")])
>>>    m2 = Metadatum("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** (*text_type, tuple*) – Name of metadatum field
- **value** (*Metadatum*) – Metadum dictionary

Returns An element of children whose index is key
Raises *TypeError* if key is not text_type or tuple of text_type
Raises *ValueError* if key and value are list and are not the same size
Example

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

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

```
>>>     a[("title", "author")] = (
>>>         Metadatum("title", [("lat", "Amores"), ("fre", "Les Amours")]),
>>>         Metadatum("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", "<M
```

__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"])
```

Variables
- ***EXPORT_TO*** – List of exportable supported formats
- ***DEFAULT_EXPORT*** – Default export (CTS XML Inventory)

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

keys()
List of keys available

Returns List of metadatum keys

class MyCapytain.common.metadata.**Metadatum**(*name*, *children=None*, *namespace=None*)
Metadatum object represent a single field of metadata

Parameters

- **name** (*text_type*) – Name of the field
- **children** (*List*) – List of tuples, where first element is the key, and second the value
- **namespace** (*Namespace*) – Object representing a namespace

Example

```
>>> a = Metadatum("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** (*text_type, 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("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** (*text_type, list, tuple*) – Index key(s) for the metadata
- **value** (*text_type, list, tuple*) – Values for the metadata

Returns An element of children whose index is key

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

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

namespace
Namespace of the metadata entry

setDefault(*key*)
Set a default key when a field does not exist

Parameters **key** (*text_type*) – An existing key of the instance

Returns Default key

Raises *ValueError* If key is not registered

Example

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

Utilities

class MyCapytain.common.utils.**OrderedDefaultDict**(*default_factory=None*, **args*, ***kwargs*)
Extension of Default Dict that makes an OrderedDefaultDict

Parameters **default_factory** – Default class to initiate

MyCapytain.common.utils.**copyNode**(*node*, *children=False*, *parent=False*)
Copy an XML Node

Parameters

- **node** – Etree Node
- **children** – Copy children nodes is set to True
- **parent** – Append copied node to parent if given

Returns New Element

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**()
Helper to create a nested ordered default dictionary

Rtype OrderedDefaultDict

Returns Nested Ordered Default Dictionary instance

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** (*text_type*) – A string to change

Return type text_type

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*, *objectify=True*)
Parse xml

Parameters **xml** (*Union[text_type, lxml.etree._Element]*) – XML element

Return type lxml.etree._Element

Returns An element object

Raises TypeError if element is not in accepted type

3.5.2 API Retrievers

Module endpoints contains prototypes and implementation of retrievers in MyCapytain

CTS 5 API

class MyCapytain.retrievers.cts5.**CTS**(*endpoint*, *inventory=None*)
Bases: MyCapytain.retrievers.prototypes.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*, *urn=None*)
Retrieve the inventory information of an API

Parameters

- **inventory** (text) – Name of the inventory
- **urn** (*str*) – URN to filter with

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

getMetadata(*objectId=None*, ***filters*)
Request metadata about a text or a collection

Parameters

- **objectId** – Filter for some object identifier
- **filters** – Kwargs parameters. URN and Inv are available

Returns GetCapabilities CTS API request response

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 information 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

getReffs(*textId*, *level=1*, *subreference=None*)
Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **level** (*int*) – Depth for retrieval
- **subreference** (*str*) – Passage Reference

Returns List of references

Return type [str]

getSiblings(*textId*, *subreference*)
Retrieve the siblings of a textual node

Parameters

- **textId** – Text Identifier
- **reference** – Passage Reference

Returns GetPrevNextUrn request response from the endpoint

getTextualNode(*textId*, *subreference=None*, *prevnext=False*, *metadata=False*)
Retrieve a text node from the API

Parameters

- **textId** – Text Identifier
- **subreference** – Passage Reference
- **prevnext** – Retrieve graph representing previous and next passage
- **metadata** – Retrieve metadata about the passage and the text

Returns GetPassage or GetPassagePlus CTS API request response

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.prototypes.**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.prototypes.**CTS**(*endpoint*)
Bases: MyCapytain.retrievers.prototypes.CitableTextServiceRetriever

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

getMetadata(*objectId=None*, ***filters*)
Request metadata about a text or a collection

Parameters

- **objectId** – Text Identifier
- **filters** – Kwargs parameters. URN and Inv are available

Returns Metadata of text from an API or the likes as bytes

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

getReffs(*textId*, *level=1*, *subreference=None*)
Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **level** (*int*) – Depth for retrieval
- **subreference** (*str*) – Passage Reference

Returns List of references

Return type [str]

getSiblings(*textId*, *subreference*)
Retrieve the siblings of a textual node

Parameters

- **textId** – Text Identifier
- **subreference** – Passage Reference

Returns Siblings references from an API or the likes as bytes

getTextualNode(*textId*, *subreference=None*, *prevnext=False*, *metadata=False*)
Retrieve a text node from the API

Parameters

- **textId** – Text Identifier
- **subreference** – Passage Reference
- **prevnext** – Retrieve graph representing previous and next passage
- **metadata** – Retrieve metadata about the passage and the text

Returns Text of a Passage from an API or the likes as bytes

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

class MyCapytain.retrievers.prototypes.**CitableTextServiceRetriever**(*endpoint*)
Bases: MyCapytain.retrievers.prototypes.API

Citable Text Service retrievers should have at least have some of the following properties

getMetadata(*objectId=None*, ***filters*)
Request metadata about a text or a collection

Parameters

- **objectId** – Text Identifier
- **filters** – Kwargs parameters. URN and Inv are available

Returns Metadata of text from an API or the likes as bytes

getReffs(*textId*, *level=1*, *subreference=None*)
Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **level** (*int*) – Depth for retrieval
- **subreference** (*str*) – Passage Reference

Returns List of references

Return type [str]

getSiblings(*textId*, *subreference*)
Retrieve the siblings of a textual node

Parameters

- **textId** – Text Identifier
- **subreference** – Passage Reference

Returns Siblings references from an API or the likes as bytes

getTextualNode(*textId*, *subreference=None*, *prevnext=False*, *metadata=False*)
Retrieve a text node from the API

Parameters

- **textId** – Text Identifier
- **subreference** – Passage Reference
- **prevnext** – Retrieve graph representing previous and next passage
- **metadata** – Retrieve metadata about the passage and the text

Returns Text of a Passage from an API or the likes as bytes

3.5.3 Resolvers

Remote CTS API

class MyCapytain.resolvers.cts.api.**HttpCTSResolver**(*endpoint*)

HttpCTSResolver provide a resolver for CTS API http endpoint.

Parameters **endpoint** (*CTS*) – CTS API Retriever

Variables ***endpoint*** – CTS API Retriever

endpoint

CTS Endpoint of the resolver

Returns CTS Endpoint

Return type CTS

getMetadata(*objectId=None*, ***filters*)

Request metadata about a text or a collection

Parameters

- **objectId** (*str*) – Object Identifier to filter on
- **filters** (*dict*) – Kwargs parameters.

Returns Collection

getReffs(*textId*, *level=1*, *subreference=None*)

Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **level** (*int*) – Depth for retrieval
- **subreference** (*str*) – Passage Reference

Returns List of references

Return type [str]

getSiblings(*textId*, *subreference*)

Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **subreference** (*str*) – Passage Reference

Returns Tuple of references

Return type (str, str)

getTextualNode(*textId*, *subreference=None*, *prevnext=False*, *metadata=False*)

Retrieve a text node from the API

Parameters

- **textId** (*str*) – Text Identifier
- **subreference** (*str*) – Passage Reference
- **prevnext** (*boolean*) – Retrieve graph representing previous and next passage

- **metadata** (*boolean*) – Retrieve metadata about the passage and the text

Returns Passage

Return type *Passage*

Local CapiTainS Guidelines CTS Resolver

class MyCapytain.resolvers.cts.local.**CTSCapitainsLocalResolver**(*resource*, *name=None*, *logger=None*)

XML Folder Based resolver. Text and metadata resolver based on local directories

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
- **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
- **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

TEXT_CLASS

alias of Text

getMetadata(*objectId=None*, ***filters*)

Request metadata about a text or a collection

Parameters

- **objectId** (*str*) – Object Identifier to filter on
- **filters** (*dict*) – Kwargs parameters.

Returns Collection

getReffs(*textId*, *level=1*, *subreference=None*)

Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **level** (*int*) – Depth for retrieval
- **subreference** (*str*) – Passage Reference

Returns List of references

Return type [str]

getSiblings(*textId*, *subreference*)

Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **subreference** (*str*) – Passage Reference

Returns Tuple of references

Return type (str, str)

getTextualNode(*textId*, *subreference=None*, *prevnext=False*, *metadata=False*)
Retrieve a text node from the API

Parameters

- **textId** (*str*) – Text Identifier
- **subreference** (*str*) – Passage Reference
- **prevnext** (*boolean*) – Retrieve graph representing previous and next passage
- **metadata** (*boolean*) – Retrieve metadata about the passage and the text

Returns Passage

Return type *Passage*

static pagination(*page*, *limit*, *length*)
Help for pagination :param page: Provided Page :param limit: Number of item to show :param length: Length of the list to paginate :return: (Start Index, End Index, Page Number, Item Count)

parse(*resource*)
Parse a list of directories ans :param resource: List of folders :param cache: Auto cache the results :return: An inventory resource and a list of Text metadata-objects

xmlparse(*file*)
Parse a XML file :param file: Opened File :return: Tree

Prototypes

class MyCapytain.resolvers.prototypes.**Resolver**
Resolver provide a native python API which returns python objects.

Initiation of resolvers are dependent on the implementation of the prototype

getMetadata(*objectId=None*, ***filters*)
Request metadata about a text or a collection

Parameters

- **objectId** (*str*) – Object Identifier to filter on
- **filters** (*dict*) – Kwargs parameters.

Returns Collection

getReffs(*textId*, *level=1*, *subreference=None*)
Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **level** (*int*) – Depth for retrieval
- **subreference** (*str*) – Passage Reference

Returns List of references

Return type [str]

getSiblings(*textId*, *subreference*)
Retrieve the siblings of a textual node

Parameters

- **textId** (*str*) – Text Identifier
- **subreference** (*str*) – Passage Reference

Returns Tuple of references

Return type (str, str)

getTextualNode(*textId*, *subreference=None*, *prevnext=False*, *metadata=False*)
Retrieve a text node from the API

Parameters

- **textId** (*str*) – Text Identifier
- **subreference** (*str*) – Passage Reference
- **prevnext** (*boolean*) – Retrieve graph representing previous and next passage
- **metadata** (*boolean*) – Retrieve metadata about the passage and the text

Returns Passage

Return type *Passage*

3.5.4 Texts and inventories

Text

TEI based texts

class MyCapytain.resources.texts.encodings.**TEIResource**(*resource*, ***kwargs*)
Bases: MyCapytain.resources.prototypes.text.InteractiveTextualNode

TEI Encoded Resource

Parameters **resource** (*Union[str,_Element]*) – XML Resource that needs to be parsed into a Passage/Text

Variables

- ***EXPORT_TO*** – List of exportable supported formats
- ***DEFAULT_EXPORT*** – Default export (Plain/Text)

DEFAULT_EXPORT = 'text/plain'

EXPORT_TO = ['python/lxml', 'text/xml', 'python/NestedDict', 'text/plain']

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference*)
Retrieve a passage and store it in the object

Parameters **subreference** (*str or Node or Reference*) – Reference of the passage to retrieve

Return type TextualNode

Returns Object representing the passage

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

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Last child of current Passage

Return type str

Returns Last passage Node representation

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Next Node (Sibling)

Return type str

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Get Previous Passage

Return type *Passage*

prevId
Previous Node (Sibling)

Return type str

siblingsId
Siblings Node

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

xml
XML Representation of the Passage

Return type lxml.etree._Element

Returns XML element representing the passage

Locally read text

class MyCapytain.resources.texts.locals.tei.**Text**(*urn=None*, *citation=None*, *resource=None*)

Bases: MyCapytain.resources.texts.locals.tei.__SharedMethods__, *MyCapytain.resources.texts.encodings.TEIResource*, MyCapytain.resources.prototypes.text.C

Implementation of CTS tools for local files

Parameters

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

Variables **resource** – lxml

DEFAULT_EXPORT = 'text/plain'

EXPORT_TO = ['python/lxml', 'text/xml', 'python/NestedDict', 'text/plain']

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getLabel()
Retrieve metadata about the text

Return type Collection

Returns Retrieve Label informations in a Collection format

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type List.basestring

Returns List of levels

getTextualNode(*subreference=None*, *simple=False*)
Finds a passage in the current text

Parameters

- **subreference** (*Union[list, Reference]*) – Identifier of the subreference / passages
- **simple** (*boolean*) – If set to true, retrieves nodes up to the given one, cleaning non required siblings.

Return type Passage, ContextPassage

Returns Asked passage

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 ?

---

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Last child of current Passage

Return type str

Returns Last passage Node representation

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Next Node (Sibling)

Return type str

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Get Previous Passage

Return type *Passage*

prevId
Previous Node (Sibling)

Return type str

reffs
Get all valid reffs for every part of the CitableText

Return type [str]

siblingsId
Siblings Node

Return type (str, str)

test()
Parse the object and generate the children

text
String representation of the text

Returns String representation of the text

Return type text_type

textObject
Textual Object with full capacities (Unlike Simple Passage)

Return type Text, Passage

Returns Textual Object with full capacities (Unlike Simple Passage)

tostring(**args*, ***kwargs*)
Transform the Passage in XML string

Parameters

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

Returns

urn
URN Identifier of the object

Return type *URN*

xml
XML Representation of the Passage

Return type lxml.etree._Element

Returns XML element representing the passage

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)

class MyCapytain.resources.texts.locals.tei.**Passage**(*reference*, *urn=None*, *citation=None*, *resource=None*, *text=None*)

Bases: MyCapytain.resources.texts.locals.tei.__SharedMethods__, *MyCapytain.resources.texts.encodings.TEIResource*, MyCapytain.resources.prototypes.text.P

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

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

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*

DEFAULT_EXPORT = 'text/plain'

EXPORT_TO = ['python/lxml', 'text/xml', 'python/NestedDict', 'text/plain']

about

Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Children of the passage

Return type None, Reference

Returns Dictionary of chidren, where key are subreferences

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getLabel()
Retrieve metadata about the text

Return type Collection

Returns Retrieve Label informations in a Collection format

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type List.basestring

Returns List of levels

getTextualNode(*subreference=None*, **args*, ***kwargs*)

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 ?

---

id

Identifier of the text

Returns Identifier of the text

Return type text_type

last

Last Passage

Return type *Passage*

lastId

Last child of current Passage

Return type str

Returns Last passage Node representation

metadata

Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next

Next Passage (Interactive Passage)

nextId

Next passage

Returns Next passage at same level

Return type None, Reference

parent

Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Previous Passage (Interactive Passage)

prevId
Get the Previous passage reference

Returns Previous passage reference at the same level

Return type None, Reference

reference
Reference of the object

siblingsId
Siblings Identifiers of the passage

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

textObject
Text Object. Required for NextPrev

Return type *Text*

tostring(**args*, ***kwargs*)
Transform the Passage in XML string

Parameters

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

Returns

urn
URN Identifier of the object

Return type *URN*

xml
XML Representation of the Passage

Return type lxml.etree._Element

Returns XML element representing the passage

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)

class MyCapytain.resources.texts.locals.tei.**__SimplePassage__**(*resource*, *reference*, *citation*, *text*, *urn=None*)

Bases: MyCapytain.resources.texts.locals.tei.__SharedMethods__, *MyCapytain.resources.texts.encodings.TEIResource*, MyCapytain.resources.prototypes.text.P

Passage for simple and quick parsing of texts

Parameters

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

about

Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds

Children of the passage

Return type None, Reference

Returns Dictionary of chidren, where key are subreferences

children

Children Passages

Return type iterator(Passage)

citation

Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

depth

Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)

Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getLabel()
Retrieve metadata about the text

Return type Collection

Returns Retrieve Label informations in a Collection format

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type List.basestring

Returns List of levels

getTextualNode(*subreference=None*)
Special GetPassage implementation for SimplePassage (Simple is True by default)

Parameters **subreference** –

Returns

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 ?

---

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Last child of current Passage

Return type str

Returns Last passage Node representation

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Next passage

Returns Next passage at same level

Return type None, Reference

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Get Previous Passage

Return type *Passage*

prevId
Get the Previous passage reference

Returns Previous passage reference at the same level

Return type None, Reference

reference
URN Passage Reference

Returns Reference

Return type *Reference*

siblingsId
Siblings Identifiers of the passage

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

textObject
Text Object. Required for NextPrev

Return type *Text*

tostring(**args*, ***kwargs*)
Transform the Passage in XML string

Parameters

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

Returns

urn
URN Identifier of the object

Return type *URN*

xml
XML Representation of the Passage

Return type lxml.etree._Element

Returns XML element representing the passage

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)

CTS API Texts

Formerly MyCapytain.resources.texts.api (< 2.0.0)

class MyCapytain.resources.texts.api.cts.**Text**(*urn*, *retriever*, *citation=None*, ***kwargs*)
Bases: MyCapytain.resources.texts.api.cts.__SharedMethod__, MyCapytain.resources.prototypes.text.CitableText

API Text object

Parameters

- **urn** (*Union[URN, str, unicode]*) – A URN identifier
- **resource** (*CitableTextServiceRetriever*) – An API endpoint
- **citation** (Citation) – Citation for children level
- **id** (*List*) – Identifier of the subreference without URN informations

DEFAULT_EXPORT = None

DEFAULT_LANG = 'eng'

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the current opbject

Returns Int representation of the depth based on URN information

Return type int

export(*output='text/plain'*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses Mimetypes)
- **exclude** (*[str]*) – Informations to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
Children passage

Return type str

Returns First children of the graph. Shortcut to self.graph.children[0]

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 str

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

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

Parameters **reference** (*Reference or List of text_type*) – 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** (*Union[Reference, str]*) – Reference from which to find siblings

Returns (Previous Passage Reference,Next Passage Reference)

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference=None*)
Retrieve a passage and store it in the object

Parameters **subreference** (*Union[Reference, URN, str, list]*) – Reference of the passage (Note : if given a list, this should be a list of string that compose the reference)

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, CitableText 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

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Children passage

Return type str

Returns First children of the graph. Shortcut to self.graph.children[0]

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next

nextId

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev

prevId

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 (str, str)

reffs
Get all valid reffs for every part of the CitableText

Return type MyCapytain.resources.texts.tei.Citation

retriever
Retriever object used to query for more data

Return type CitableTextServiceRetriever

siblingsId

text
String representation of the text

Returns String representation of the text

Return type text_type

urn
URN Identifier of the object

Return type *URN*

class MyCapytain.resources.texts.api.cts.**Passage**(*urn*, *resource*, **args*, ***kwargs*)
Bases: MyCapytain.resources.texts.api.cts.__SharedMethod__, MyCapytain.resources.prototypes.text.Passage, *MyCapytain.resources.texts.encodings.TEIRe*

Passage representing

Parameters

- **urn** –
- **resource** –
- **retriever** –
- **args** –
- **kwargs** –

DEFAULT_EXPORT = 'text/plain'

EXPORT_TO = ['python/lxml', 'text/xml', 'python/NestedDict', 'text/plain']

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the current opbject

Returns Int representation of the depth based on URN information

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
Children passage

Return type str

Returns First children of the graph. Shortcut to self.graph.children[0]

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 str

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

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

Parameters **reference** (*Reference or List of text_type*) – 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** (*Union[Reference, str]*) – Reference from which to find siblings

Returns (Previous Passage Reference,Next Passage Reference)

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference=None*)
Retrieve a passage and store it in the object

Parameters **subreference** (*Union[Reference, URN, str, list]*) – Reference of the passage (Note : if given a list, this should be a list of string that compose the reference)

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, CitableText 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

id

last
Last Passage

Return type *Passage*

lastId
Children passage

Return type str

Returns First children of the graph. Shortcut to self.graph.children[0]

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Shortcut for getting the following passage identifier

Return type *Reference*

Returns Following passage reference

parent
Parent Passage

Return type *Passage*

parentId
Shortcut for getting the parent passage identifier

Return type *Reference*

Returns Following passage reference

prev
Get Previous Passage

Return type *Passage*

prevId
Previous passage Identifier

Return type *Passage*

Returns Previous passage at same level

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 (str, str)

reference

retriever
Retriever object used to query for more data

Return type CitableTextServiceRetriever

siblingsId
Shortcut for getting the previous and next passage identifier

Return type *Reference*

Returns Following passage reference

text
String representation of the text

Returns String representation of the text

Return type text_type

urn
URN Identifier of the object

Return type *URN*

xml
XML Representation of the Passage

Return type lxml.etree._Element

Returns XML element representing the passage

Collections

Metadata

class MyCapytain.resources.prototypes.metadata.**Collection**
Bases: MyCapytain.common.constants.Exportable

Collection represents any resource's metadata. It has members and parents

Variables

- **properties** – Properties of the collection
- **parents** – Parent of the node from the direct parent to the highest ascendant
- ***metadata*** – Metadata
- **DC_TITLE_KEY** – Key representing the object title in the Metadata property

DC_TITLE_KEY = None

DEFAULT_EXPORT = None

EXPORT_TO = ['application/ld+json:DTS/NoParents', 'application/ld+json:DTS']

TYPE_URI = 'http://w3id.org/dts-ontology/collection'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id
Identifier of the collection item

Return type str

members
Children of the collection's item

Return type [Collection]

readable
Readable property should return elements where the element can be queried for getPassage / getReffs

readableDescendants
List of element available which are readable

Return type [Collection]

title
Title of the collection Item

Return type Metadatum

CTS inventory

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

Bases: *MyCapytain.common.reference.Citation*

Citation XML implementation for TextInventory

child
Child of a citation

Type Citation or None

Example Citation.name==poem would have a child Citation.name==line

escape = re.compile('(")')

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']
```

static ingest(*resource*, *element=None*, *xpath='ti:citation'*)
Ingest xml to create a citation

Parameters

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

Returns Citation

isEmpty()
Check if the citation has not been set

Returns True if nothing was setup

Return type bool

name
Type of the citation represented

Type text_type

Example Book, Chapter, Textpart, Section, Poem...

refsDecl
ResfDecl expression of the citation scheme

Return type str

Example /tei:TEI/tei:text/tei:body/tei:div//tei:l[@n='$1']

scope
TextInventory scope property of a citation (ie. identifier of all element but the last of the citation)

Type basestring

Example /tei:TEI/tei:text/tei:body/tei:div

xpath
TextInventory xpath property of a citation (ie. identifier of the last element of the citation)

Type basestring

Example //tei:l[@n="?"]

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

class MyCapytain.resources.collections.cts.**Text**(***kwargs*)
Bases: MyCapytain.resources.prototypes.cts.inventory.Text

Represents a CTS Text

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = 'label'

DEFAULT_EXPORT = 'python/lxml'

EXPORT_TO = ['Capitains/ReadableText', 'python/lxml', 'text/xml:CTS']

TEXT_URI
Ontology URI of the text

Returns CTS Ontology Edition or Translation object

Return type str

TYPE_URI = 'http://w3id.org/dts-ontology/collection'

descendants
List of descendants

Return type *list*

editions()
Get all editions of the texts

Returns List of editions

Return type [Text]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id

members

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

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

Returns None

readable
Readable property should return elements where the element can be queried for getPassage / getReffs

Return type bool

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

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

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

Returns

class MyCapytain.resources.collections.cts.**TextGroup**(***kwargs*)
Bases: MyCapytain.resources.prototypes.cts.inventory.TextGroup

Represents a CTS Textgroup in XML

Variables

- ***EXPORT_TO*** – List of exportable supported formats
- ***DEFAULT_EXPORT*** – Default export (CTS XML Inventory)

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = 'groupname'

DEFAULT_EXPORT = 'python/lxml'

EXPORT_TO = ['python/lxml', 'text/xml:CTS']

TYPE_URI = 'http://chs.harvard.edu/xmlns/cts/TextGroup'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id

members

parse(*resource*)
Parse a resource

Parameters

- **resource** – Element representing the textgroup
- **type** – basestring or etree._Element

readable
Readable property should return elements where the element can be queried for getPassage / getReffs

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

update(*other*)
Merge two Textgroup Objects.

•Original (left Object) keeps his parent.

•Added document merges with work if it already exists

Parameters **other** (*TextGroup*) – Textgroup object

Returns Textgroup Object

Return type TextGroup

class MyCapytain.resources.collections.cts.**TextInventory**(***kwargs*)
Bases: MyCapytain.resources.prototypes.cts.inventory.TextInventory

Represents a CTS Inventory file

Variables

- ***EXPORT_TO*** – List of exportable supported formats
- ***DEFAULT_EXPORT*** – Default export (CTS XML Inventory)

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = None

DEFAULT_EXPORT = 'python/lxml'

EXPORT_TO = ['python/lxml', 'text/xml:CTS']

TYPE_URI = 'http://chs.harvard.edu/xmlns/cts/TextInventory'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id

members

parse(*resource*)
Parse a resource

Parameters

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

readable
Readable property should return elements where the element can be queried for getPassage / getReffs

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

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

class MyCapytain.resources.collections.cts.**Work**(***kwargs*)
Bases: MyCapytain.resources.prototypes.cts.inventory.Work

Represents a CTS Textgroup in XML

Variables

- ***EXPORT_TO*** – List of exportable supported formats
- ***DEFAULT_EXPORT*** – Default export (CTS XML Inventory)

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = 'title'

DEFAULT_EXPORT = 'python/lxml'

EXPORT_TO = ['python/lxml', 'text/xml:CTS']

TYPE_URI = 'http://chs.harvard.edu/xmlns/cts/Work'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

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

Parameters **key** (*text_type*) – Language to find

Return type [Text]

Returns List of availables translations

id

members

parse(*resource*)
Parse a resource

Parameters

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

readable

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

update(*other*)
Merge two Work Objects.

•Original (left Object) keeps his parent.

•Added document overwrite text if it already exists

Parameters **other** (*Work*) – Work object

Returns Work Object

Rtype Work

MyCapytain.resources.collections.cts.**xpathDict**(*xml*, *xpath*, *children*, *parents*, ***kwargs*)
Returns a default Dict given certain information

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

CTS Inventory Prototypes

class MyCapytain.resources.prototypes.cts.inventory.**CTSCollection**(*resource=None*)
Bases: MyCapytain.resources.prototypes.metadata.Collection

Resource represents any resource from the inventory

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

Variables **CTSMODEL** – String Representation of the type of collection

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = None

DEFAULT_EXPORT = None

EXPORT_TO = ['application/ld+json:DTS/NoParents', 'application/ld+json:DTS']

TYPE_URI = 'http://w3id.org/dts-ontology/collection'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id
Identifier of the collection item

Return type str

members
Children of the collection's item

Return type [Collection]

parse(*resource*)
Parse the object resource

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

Return type List

readable
Readable property should return elements where the element can be queried for getPassage / getReffs

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

MyCapytain.resources.prototypes.cts.inventory.**Edition**(*resource=None*, *urn=None*, *parents=None*)
Represents a CTS Edition

Parameters

- **resource** (*Any*) – Resource representing the TextInventory
- **urn** (*str*) – Identifier of the Text
- **parents** (*[CTSCollection]*) – Item parents of the current collection

class MyCapytain.resources.prototypes.cts.inventory.**Text**(*resource=None*, *urn=None*, *parents=None*, *subtype='Edition'*)
Bases: MyCapytain.resources.prototypes.cts.inventory.CTSCollection

Represents a CTS Text

Parameters

- **resource** (*Any*) – Resource representing the TextInventory
- **urn** (URN) – Identifier of the Text
- **parents** (*[CTSCollection]*) – Item parents of the current collection
- **subtype** (*str*) – Subtype of the object (Edition, Translation)

Variables

- ***urn*** – URN Identifier
- **parents** – List of ancestors, from parent to furthest

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = 'label'

DEFAULT_EXPORT = None

EXPORT_TO = ['application/ld+json:DTS/NoParents', 'application/ld+json:DTS']

TEXT_URI
Ontology URI of the text

Returns CTS Ontology Edition or Translation object

Return type str

TYPE_URI = 'http://w3id.org/dts-ontology/collection'

descendants

editions()
Get all editions of the texts

Returns List of editions

Return type [Text]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id

members

parse(*resource*)
Parse the object resource

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

Return type List

readable

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

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

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

Returns

class MyCapytain.resources.prototypes.cts.inventory.**TextGroup**(*resource=None*, *urn=None*, *parents=None*)
Bases: MyCapytain.resources.prototypes.cts.inventory.CTSCollection

Represents a CTS Textgroup

CTS TextGroup can be added to each other which would most likely happen if you take your data from multiple API or Textual repository. This works close to dictionary update in Python. See update

Parameters

- **resource** (*Any*) – Resource representing the TextInventory
- **urn** (URN) – Identifier of the TextGroup
- **parents** (*Tuple.<TextInventory>*) – List of parents for current object

Variables

- ***urn*** – URN Identifier
- **parents** – List of ancestors, from parent to furthest

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = 'groupname'

DEFAULT_EXPORT = None

EXPORT_TO = ['application/ld+json:DTS/NoParents', 'application/ld+json:DTS']

TYPE_URI = 'http://chs.harvard.edu/xmlns/cts/TextGroup'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id

members

parse(*resource*)
Parse the object resource

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

Return type List

readable
Readable property should return elements where the element can be queried for getPassage / getReffs

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

update(*other*)
Merge two Textgroup Objects.

•Original (left Object) keeps his parent.

•Added document merges with work if it already exists

Parameters **other** (*TextGroup*) – Textgroup object

Returns Textgroup Object

Return type TextGroup

class MyCapytain.resources.prototypes.cts.inventory.**TextInventory**(*resource=None*, *name=None*)
Bases: MyCapytain.resources.prototypes.cts.inventory.CTSCollection

Initiate a TextInventory resource

Parameters

- **resource** (*Any*) – Resource representing the TextInventory
- **id** (*str*) – Identifier of the TextInventory

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = None

DEFAULT_EXPORT = None

EXPORT_TO = ['application/ld+json:DTS/NoParents', 'application/ld+json:DTS']

TYPE_URI = 'http://chs.harvard.edu/xmlns/cts/TextInventory'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export(*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id

members

parse(*resource*)
Parse the object resource

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

Return type List

readable
Readable property should return elements where the element can be queried for getPassage / getReffs

readableDescendants
List of element available which are readable

Return type [Collection]

setResource(*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

MyCapytain.resources.prototypes.cts.inventory.**Translation**(*resource=None*, *urn=None*, *parents=None*)
Represents a CTS Translation

Parameters

- **resource** (*Any*) – Resource representing the TextInventory
- **urn** (*str*) – Identifier of the Text
- **parents** (*[CTSCollection]*) – Item parents of the current collection

class MyCapytain.resources.prototypes.cts.inventory.**Work**(*resource=None*, *urn=None*, *parents=None*)
Bases: MyCapytain.resources.prototypes.cts.inventory.CTSCollection

Represents a CTS Work

CTS Work can be added to each other which would most likely happen if you take your data from multiple API or Textual repository. This works close to dictionary update in Python. See update

Parameters

- **resource** (*Any*) – Resource representing the TextInventory
- **urn** (URN) – Identifier of the Work
- **parents** (*Tuple.<TextInventory>*) – List of parents for current object

Variables

- ***urn*** – URN Identifier
- **parents** – List of ancestors, from parent to furthest

CTSMODEL = 'CTSCollection'

DC_TITLE_KEY = 'title'

DEFAULT_EXPORT = None

EXPORT_TO = ['application/ld+json:DTS/NoParents', 'application/ld+json:DTS']

TYPE_URI = 'http://chs.harvard.edu/xmlns/cts/Work'

descendants
Any descendant (no max level) of the collection's item

Return type [Collection]

export (*output=None*, ***kwargs*)
Export the collection item in the Mimetype required.

Parameters **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.utils.Mimetypes)

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

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

Parameters **key** (*text_type*) – Language to find

Return type [Text]

Returns List of availables translations

id

members

parse (*resource*)
Parse the object resource

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

Return type List

readable

readableDescendants
List of element available which are readable

Return type [Collection]

setResource (*resource*)
Set the object property resource

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

Return type Any

Returns Input resource

title
Title of the collection Item

Return type Metadatum

update (*other*)
Merge two Work Objects.

- Original (left Object) keeps his parent.
- Added document overwrite text if it already exists

Parameters **other** (*Work*) – Work object

Returns Work Object

Rtype Work

Text Prototypes

class MyCapytain.resources.prototypes.text.**CTSNode**(*urn=None*, ***kwargs*)
Bases: MyCapytain.resources.prototypes.text.InteractiveTextualNode

Initiate a Resource object

Parameters

- **urn** (URN) – A URN identifier
- **metadata** (*Collection*) – Collection Information about the Item
- **citation** (Citation) – Citation system of the text
- **children** (*[str]*) – Current node Children's Identifier
- **parent** (*str*) – Parent of the current node
- **siblings** (*str*) – Previous and next node of the current node
- **depth** (*int*) – Depth of the node in the global hierarchy of the text tree
- **resource** – Resource used to navigate through the textual graph

Variables ***default_exclude*** – Default exclude for exports

DEFAULT_EXPORT = None

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getLabel()
Retrieve metadata about the text

Return type Collection

Returns Retrieve Label informations in a Collection format

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference*)
Retrieve a passage and store it in the object

Parameters **subreference** (*str or Node or Reference*) – Reference of the passage to retrieve

Return type TextualNode

Returns Object representing the passage

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

getValidReff(*level=1*, *reference=None*)
Given a resource, CitableText 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.text_type

Returns List of levels

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Last child of current Passage

Return type str

Returns Last passage Node representation

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Next Node (Sibling)

Return type str

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Get Previous Passage

Return type *Passage*

prevId
Previous Node (Sibling)

Return type str

siblingsId
Siblings Node

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

urn
URN Identifier of the object

Return type *URN*

class MyCapytain.resources.prototypes.text.**CitableText**(*citation=None*, *metadata=None*, ***kwargs*)
Bases: MyCapytain.resources.prototypes.text.CTSNode

A CTS CitableText

DEFAULT_EXPORT = None

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getLabel()
Retrieve metadata about the text

Return type Collection

Returns Retrieve Label informations in a Collection format

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference*)
Retrieve a passage and store it in the object

Parameters **subreference** (*str or Node or Reference*) – Reference of the passage to retrieve

Return type TextualNode

Returns Object representing the passage

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

getValidReff(*level=1*, *reference=None*)
Given a resource, CitableText 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.text_type

Returns List of levels

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Last child of current Passage

Return type str

Returns Last passage Node representation

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Next Node (Sibling)

Return type str

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Get Previous Passage

Return type *Passage*

prevId
Previous Node (Sibling)

Return type str

reffs
Get all valid reffs for every part of the CitableText

Return type [str]

siblingsId
Siblings Node

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

urn
URN Identifier of the object

Return type *URN*

class MyCapytain.resources.prototypes.text.**InteractiveTextualNode**(*identifier=None*, ***kwargs*)

Bases: MyCapytain.resources.prototypes.text.TextualGraph

Node representing a text passage.

Parameters

- **identifier** (*str*) – Identifier of the text
- **metadata** (*Collection*) – Collection Information about the Item
- **citation** (Citation) – Citation system of the text
- **children** (*[str]*) – Current node Children's Identifier
- **parent** (*str*) – Parent of the current node
- **siblings** (*str*) – Previous and next node of the current node
- **depth** (*int*) – Depth of the node in the global hierarchy of the text tree
- **resource** – Resource used to navigate through the textual graph

Variables ***default_exclude*** – Default exclude for exports

DEFAULT_EXPORT = None

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference*)
Retrieve a passage and store it in the object

Parameters **subreference** (*str or Node or Reference*) – Reference of the passage to retrieve

Return type TextualNode

Returns Object representing the passage

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

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Last child of current Passage

Return type str

Returns Last passage Node representation

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Next Node (Sibling)

Return type str

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Get Previous Passage

Return type *Passage*

prevId
Previous Node (Sibling)

Return type str

siblingsId
Siblings Node

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

class MyCapytain.resources.prototypes.text.**Passage**(***kwargs*)
Bases: MyCapytain.resources.prototypes.text.CTSNode

Passage objects possess metadata informations

Parameters

- **urn** (URN) – A URN identifier
- **metadata** (*Collection*) – Collection Information about the Item
- **citation** (Citation) – Citation system of the text
- **children** (*[str]*) – Current node Children's Identifier
- **parent** (*str*) – Parent of the current node
- **siblings** (*str*) – Previous and next node of the current node
- **depth** (*int*) – Depth of the node in the global hierarchy of the text tree
- **resource** – Resource used to navigate through the textual graph

Variables ***default_exclude*** – Default exclude for exports

DEFAULT_EXPORT = None

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Identifiers of children

Returns Identifiers of children

Return type [str]

children
Children Passages

Return type iterator(Passage)

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

first
First Passage

Return type *Passage*

firstId
First child of current Passage

Return type str

Returns First passage node Information

getLabel()
Retrieve metadata about the text

Return type Collection

Returns Retrieve Label informations in a Collection format

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference*)
Retrieve a passage and store it in the object

Parameters **subreference** (*str or Node or Reference*) – Reference of the passage to retrieve

Return type TextualNode

Returns Object representing the passage

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

getValidReff(*level=1*, *reference=None*)
Given a resource, CitableText 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.text_type

Returns List of levels

id
Identifier of the text

Returns Identifier of the text

Return type text_type

last
Last Passage

Return type *Passage*

lastId
Last child of current Passage

Return type str

Returns Last passage Node representation

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

next
Get Next Passage

Return type *Passage*

nextId
Next Node (Sibling)

Return type str

parent
Parent Passage

Return type *Passage*

parentId
Parent Node

Return type str

prev
Get Previous Passage

Return type *Passage*

prevId
Previous Node (Sibling)

Return type str

reference

siblingsId
Siblings Node

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

urn
URN Identifier of the object

Return type *URN*

class MyCapytain.resources.prototypes.text.**TextualElement**(*identifier=None*, *metadata=None*)

Bases: MyCapytain.common.constants.Exportable

Node representing a text passage.

Parameters

- **identifier** (*str*) – Identifier of the text
- **metadata** (*Collection*) – Collection Information about the Item

Variables ***default_exclude*** – Default exclude for exports

DEFAULT_EXPORT = None

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

default_exclude = []

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

id
Identifier of the text

Returns Identifier of the text

Return type text_type

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

text
String representation of the text

Returns String representation of the text

Return type text_type

class MyCapytain.resources.prototypes.text.**TextualGraph**(*identifier=None*, ***kwargs*)
Bases: MyCapytain.resources.prototypes.text.TextualNode

Node representing a text passage.

Parameters

- **identifier** (*str*) – Identifier of the text
- **metadata** (*Collection*) – Collection Information about the Item
- **citation** (Citation) – Citation system of the text
- **children** (*[str]*) – Current node Children's Identifier
- **parent** (*str*) – Parent of the current node
- **siblings** (*str*) – Previous and next node of the current node
- **depth** (*int*) – Depth of the node in the global hierarchy of the text tree
- **resource** – Resource used to navigate through the textual graph

Variables ***default_exclude*** – Default exclude for exports

DEFAULT_EXPORT = None

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Children Node

Return type [str]

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

firstId
First child Node

Return type str

getReffs(*level=1*, *subreference=None*)
Reference available at a given level

Parameters

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

Return type [text_type]

Returns List of levels

getTextualNode(*subreference*)
Retrieve a passage and store it in the object

Parameters **subreference** (*str or Node or Reference*) – Reference of the passage to retrieve

Return type TextualNode

Returns Object representing the passage

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

id
Identifier of the text

Returns Identifier of the text

Return type text_type

lastId
Last child Node

Return type str

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

nextId
Next Node (Sibling)

Return type str

parentId
Parent Node

Return type str

prevId
Previous Node (Sibling)

Return type str

siblingsId
Siblings Node

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

class MyCapytain.resources.prototypes.text.**TextualNode**(*identifier=None*, *citation=None*, ***kwargs*)
Bases: MyCapytain.resources.prototypes.text.TextualElement, *MyCapytain.common.reference.NodeId*

Node representing a text passage.

Parameters

- **identifier** (*str*) – Identifier of the text
- **metadata** (*Collection*) – Collection Information about the Item
- **citation** (Citation) – Citation system of the text
- **children** (*[str]*) – Current node Children's Identifier

- **parent** (*str*) – Parent of the current node
- **siblings** (*str*) – Previous and next node of the current node
- **depth** (*int*) – Depth of the node in the global hierarchy of the text tree

Variables ***default_exclude*** – Default exclude for exports

DEFAULT_EXPORT = None

EXPORT_TO = []

about
Metadata information about the text

Returns Collection object with metadata about the text

Rtype Collection

childIds
Children Node

Return type [str]

citation
Citation Object of the Text

Returns Citation Object of the Text

Return type *Citation*

default_exclude = []

depth
Depth of the node in the global hierarchy of the text tree

Return type int

export(*output=None*, *exclude=None*, ***kwargs*)
Export the collection item in the Mimetype required.

..note:: If current implementation does not have special mimetypes, reuses default_export method

Parameters

- **output** (*str*) – Mimetype to export to (Uses MyCapytain.common.constants.Mimetypes)
- **exclude** (*[str]*) – Information to exclude. Specific to implementations

Returns Object using a different representation

export_capacities
List Mimetypes that current object can export to

firstId
First child Node

Return type str

id
Identifier of the text

Returns Identifier of the text

Return type text_type

lastId
Last child Node

Return type str

metadata
Metadata information about the text

Returns Collection object with metadata about the text

Return type Metadata

nextId
Next Node (Sibling)

Return type str

parentId
Parent Node

Return type str

prevId
Previous Node (Sibling)

Return type str

siblingsId
Siblings Node

Return type (str, str)

text
String representation of the text

Returns String representation of the text

Return type text_type

3.6 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

m

Symbols

A

C

G

H

I

L

M

N

P

R

S

T

U

V

W

X