
HookTest Documentation

Release 0.1.0

Thibault Clérice

Jul 27, 2018

Contents

1	What are HookTest and CapiTainS	3
1.1	Installation Instructions	3
2	How to use locally	5
3	Running HookTest on Travis CI	7
4	Licenses	11
4.1	TEI and EpiDoc Schema	11

docs/_static/images/header.png

- *What are HookTest and CapiTainS*
- *How to use locally*
- *Running HookTest on Travis CI*
- *Licenses*

What are HookTest and CapiTainS

```
docs/_static/images/capitains.png
```

CapiTainS provides resources for people to publish, use and reuse texts with/for standards API. Capitains HookTest is a python library and commandline tool for testing Capitains textual repositories with their metadata.

1.1 Installation Instructions

To install it, simply do : `pip3 install HookTest` or

```
git clone https://github.com/Capitains/HookTest.git
cd HookTest
python3 setup.py install
```

From there, you will be able to call it in your python scripts with `import HookTest` or you can use it in your terminal session

Be careful, as Capitains requires java for Schematron and RelaxNG tests

CHAPTER 2

How to use locally

The command is run with `hooktest [-h] [-w WORKERS] [-s SCHEME] [-v] [-j JSON] [-c] [-p PING] [-f FINDER] path` where Path is the path to the containing repository (in which there is a folder data/)

Parameter in console	Detail about the Parameter
-h, -help	show this help message and exit
-w WORKERS, -workers WORKERS	Number of workers to be used
-s SCHEME, -scheme SCHEME	Possible Values: <ul style="list-style-type: none"> • “tei”: Use the most recent TEI-ALL DTD • “epidoc”: Use the most recent epiDoc DTD • “ignore”: Perform no schema validation • “auto” (Default) - Automatically detect the RNG to use from the xml-model declaration in each individual XML file. If the reference is to a remote URL, the file will be downloaded and used. • <filepath>: If a file path is given, this should refer to a local RNG file that should be used to check all text documents
-guidelines	Possible Values: <ul style="list-style-type: none"> • “2.tei” (Default) - Will use version 2.0 of the CapiTainS guidelines for generic TEI texts. • “2.epidoc” - Will use version 2.0 of the CapiTainS guidelines for EpiDoc encoded texts. (Details at http://capitains.org/pages/guidelines#urn-information)
-v, -verbose [{0,5,7,10}]	Verbose Level <ul style="list-style-type: none"> • 0 (Default) Only show necessary Information • 5 Show duplicate or forbidden characters • 7 All of before + show failing units • 10 All available details
-j JSON, -json JSON	Save to specified json file the results
-c, -console	Print to console
-f FILTER, -filter FILTER	Filter using the last part of the URN (eg. tlg0001.tlg001, tlg0001, tlg0001.tlg001.p-grc1 for urn:cts:greekLit:tlg0001.tlg001.p-grc1)
-countword	Count words in texts passing the tests
-manifest	Produce a Manifest
-allowfailure	Returns a passing test result as long as at least one text passes

Running HookTest on Travis CI

HookTest can now be run on the Travis Continuous Integration (CI) platform. This relieves the need for HookTest user to set up their own HookTest testing server and also allows for automatic building of corpus releases after successful tests. To set up your Github CapiTainS text repository to use Travis CI, the first step is to set up your account at Travis (<https://docs.travis-ci.com/user/getting-started>). Follow step 1 and step 2 on that web page to set up your repository on Travis.

Once you have done this, you will need to add a `.travis.yml` file to root folder of your repository. (Note that the name of the file starts with a period ('.').) Use the following as a template for your own `.travis.yml` file:

```
language: python
python:
- '3.5'
install:
- pip3 install HookTest
script: hooktest --console --scheme epidoc --workers 3 --verbose 5 --manifest --
↳countword --allowfailure ./
before_deploy:
- hooktest-build --travis --txt ./
- results=$(cat manifest.txt)
- DATE=`date +%Y-%m-%d`
- git config --global user.email "builds@travis-ci.com"
- git config --global user.name "Travis CI"
- export GIT_TAG=$major_version.$minor_version.$TRAVIS_BUILD_NUMBER
- git add -A
- git tag $GIT_TAG -a -m "$DATE" -m "PASSING FILES" -m "$results"
- git push -q https://$GITPERM@github.com/YOUR_REPOSITORY_NAME --tags
- ls -R

deploy:
  provider: releases
  api_key: $GITPERM
  skip_cleanup: true
  on:
    repo: YOUR_REPOSITORY_NAME
```

(continues on next page)

(continued from previous page)

```

branch: master

env:
  global:
    major_version: 0
    minor_version: 0

```

To help you set up this file for your own repository, a line-by-line explanation follows.

```

language: python
python:
  - '3.5'
install:
  - pip3 install HookTest>=1.0.0

```

These first 5 lines are for the basic setup of HookTest on Travis. Do not change them.

```

script: hooktest --scheme epidoc --workers 3 --verbose --manifest --console --
↳countword --allowfailure ./

```

This line runs HookTest. The parameters are those described in the parameter table above. If you do not want to make a new release of your corpus unless it is 100% CapiTainS-compliant, then remove the `-allowfailure` parameter. Without this parameter, the build will fail if the corpus is not 100% compliant causing Travis to skip the build and release steps. Because of the way Travis is set up, we recommend not setting `-workers` higher than 3.

```

before_deploy:
  - hooktest-build --travis --txt ./
  - results=$(cat manifest.txt)
  - DATE=`date +%Y-%m-%d`
  - git config --global user.email "builds@travis-ci.com"
  - git config --global user.name "Travis CI"
  - export GIT_TAG=$major_version.$minor_version.$TRAVIS_BUILD_NUMBER
  - git add -A
  - git tag $GIT_TAG -a -m "$DATE" -m "PASSING FILES" -m "$results"
  - git push -q https://$GITPERM@github.com/YOUR_REPOSITORY_NAME --tags
  - ls -R

```

Once HookTest has run on Travis, if the repository is 100% CapiTainS-compliant or if the `-allowfailure` parameter was set and at least one text, along with all of its metadata files, passed, then Travis carries out the build step. Of special note here is the `hooktest-build --travis --txt ./` line. The `hooktest-build` class is designed to build the passing files in a repository into a release. To this point, it has been implemented only for Travis CI. This script basically removes all failing files from the repository. The `-txt` parameter then converts each of the passing XML text files to plain text, with each citation unit separated by two carriage returns, e.g.,:

```

Lorem ipsum dolor sit amet, consectetur adipiscing elit...

Lorem ipsum dolor sit amet, consectetur adipiscing elit...

```

Simply remove the `-txt` parameter from the `.travis.yml` file if you would prefer not to release plain text versions of your texts.

Of special note here are two things that you will need to set up yourself. The first is the environment variable `$GITPERM`. This variable should contain the value of a Github OAuth token that you have set up for your Github account. To find out how to set up such a token, see the Github documentation at <https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/>. Your OAuth token should have the `repo` scope (<https://developer.github.com/v3/oauth/#scopes>). Once you have created this token, you should define this as the

GITPERM environment variable for this repository in Travis. To do this, see the documentation here: <https://docs.travis-ci.com/user/environment-variables/#Defining-Variables-in-Repository-Settings>. Make sure that the switch for “Display value in build log” is set to off, otherwise anyone looking at your build log will be able to see your private OAuth token.

The second important change to this line is to replace the string “YOUR_REPOSITORY_NAME” with the Github user name or organization name and the repository name, e.g., “OpenGreekAndLatin/First1KGreek”. If any of these pre-deployment steps fail, then the repository will not build and release.

```
deploy:
  provider: releases
  api_key: $GITPERM
  skip_cleanup: true
  on:
    repo: YOUR_REPOSITORY_NAME
    branch: master

env:
  global:
    major_version: 0
    minor_version: 0
```

These lines define the deployment and release of your repository to Github. They will create a release on Github that has as its label the `major_version.minor_version.$TRAVIS_BUILD_NUMBER`. You should set the `major_version` and `minor_version` environment variables to match the release status of your repository.

Once you have created and tailored this `.travis.yml` file to your repository, you should then push it to your Github corpus repository. If you have set up Travis to test with repository, as described above, then Travis should read this `.travis.yml` file and automatically run HookTest and, if appropriate, build your first automatic release for the repository.

4.1 TEI and EpiDoc Schema

The TEI Schema is copyright the TEI Consortium (http://www.tei-c.org/Guidelines/access.xml#body.1_div.2). To the extent that the EpiDoc ODD and schema have been customized and amount to transformative versions of the original schema, they are copyright Gabriel Bodard and the other contributors (as listed in `tei:revisionDesc`). See `LICENSE.txt` for license details.

Contents:

4.1.1 HookTest API Documentation

Library Structure

The library is divided in three different modules : - **cmd** is the module for running HookTest in command line. - **test** is the module for the main pipeline : it takes care of finding the files to test, dispatching the test on them and interpret them - **units** is the module for each specific filetype test : it contains the logic of tests for individual files

Commands

`HookTest.cmd.parse_args(args)`

Parsing function. Written to support unit test

Parameters `args` – List of command line argument

Returns Parsed argument

`HookTest.cmd.cmd()`

Run locally the software. Should not be called outside of a python `cmd.py` call

Test Pipeline

Files Finders

class `HookTest.test.DefaultFinder` (***options*)

Finder are object used in Test to retrieve the target files of the tests

find (*directory*)

Return object to find

Parameters `directory` – Root Directory to search in

Returns Path of xml text files, Path of `__cts__.xml` files

Return type (list, list)

class `HookTest.test.FilterFinder` (*include, **options*)

FilterFinder provide a filtering capacity to DefaultFinder.

It takes an include option which takes the form of the work urn (*ie.* in `urn:cts:latinLit:phi1294.phi002.perseus-lat2` this would be `phi1294.phi002.perseus-lat2`, cut at any of the points : `phi1294`, `phi1294.phi002`, `phi1294.phi002.perseus-lat2`)

Parameters `include` (*str*) – Representation of the work urn component (might be from one member down to the version member)

find (*directory*)

Return object to find

Parameters `directory` – Root Directory to search in

Returns Path of xml text files, Path of `__cts__.xml` files

Return type (list, list)

Pipeline

class `HookTest.test.Test` (*path, workers=1, scheme='auto', verbose=0, ping=None, secret="", triggering_size=None, console=False, build_manifest=False, finder=<class 'HookTest.test.DefaultFinder'>, finderoptions=None, countwords=False, allowfailure=False, from_travis_to_hook=False, timeout=30, guidelines=None, **kwargs*)

Create a Test object

Parameters

- **path** (*str*) – Path where the test should happen
- **workers** (*str*) – Number of simultaneous workers to be used
- **scheme** (*str*) – Name of the scheme
- **verbose** (*int*) – Log also rng and unit logs details
- **ping** (*str*) – URI to ping with data
- **console** (*bool*) – If set to true, print logs to the console
- **finder** (`DefaultFinder`) – Test files retriever
- **finderoptions** (*dict*) – Dictionary of option to instantiate specific finders
- **countwords** (*bool*) – Enable counting words for text tests (False by default)

cover (*name, test, testtype=None, logs=None, additional=None*)

Given a dictionary, compute the coverage of one item

Parameters

- **name** –
- **test** (*boolean*) – Dictionary where keys represents test done on a file and value a boolean indicating passing status
- **logs** (*list*) – List of logs for one unit
- **testtype** (*str*) – the type of file tested (e.g., CTSMetadata or CTSText)

Returns Passing status

Return type dict

create_manifest ()

Creates a manifest.txt file in the source directory that contains an ordered list of passing files

directory

Directory :return: Path of the full directory :rtype: str

download ()

Information to send or print during download

end ()

Deal with end logs

find ()

Find CTS files in a directory :param directory: Path of the directory :type directory: str

Returns Path of xml text files, Path of __cts__.xml files

Return type (list, list)

flush (*stack*)

Flush the remaining logs to the endpoint

json

Get Json representation of object report

Returns JSON representing the complete test

Return type

log (*log*)

Deal with middle process situation

Parameters **log** (*UnitLog*) – Result of a test for one unit

Returns None

middle ()

to print out the results for the metadata files that failed the tests

Returns

Return type

report

Get the report of the Test :return: Report of the test :rtype: dict

run ()

Run the tests

Returns Status of the test, List of logs, Report

Return type (string, list, dict)

send (*data*)

Send data to self.ping URL

Parameters **data** – Data to send

Returns Result of request

send_to_hook_from_travis (*texts_total, texts_passing, metadata_total, metadata_passing, coverage, nodes_count, words_dict=None*)

Send data to travis

Returns Request output

stack

Get the current stack of unsent item

Returns Unset UnitLog

Return type [*UnitLog*]

start ()

Deal with the start of the process

status

Updated the status string based on available informations

Returns Status string updated

Return type str

successes

Get the number of successful tests

Returns Number of successful tests

Return type int

triggering_size

Returns

unit (*filepath*)

Do test for a file and print the results

Parameters **filepath** (*str*) – Path of the file to be tested

Returns A UnitLog

Return type *UnitLog*

HookTest.test.cmd (*console=False, **kwargs*)

Generate the complete process of Test

Parameters

- **console** (*bool*) – Print logs to console
- **kwargs** (*dict*) – Named arguments

Returns Status of the test

Models

class HookTest.test.**UnitLog** (*directory, name, units, coverage, status, testtype=None, logs=None, sent=False, additional=None*)

Model for logging information

Parameters

- **name** – Name of the tested unit
- **units** –
- **coverage** – Percentage of successful tests
- **status** – Status of the unit
- **logs** – Logs
- **sent** – Status regarding the logging
- **additional** – Additional informations. Can be used for words counting

dict

Get the dictionary version of the object

Returns Dictionary representation of the object

Return type dict

Test units

class HookTest.units.**TESTUnit** (*path*)

TestUnit Metaclass

Parameters **path** – path of the current file

parsable()

Check and parse the xml file

Returns Indicator of success and messages

Return type boolean

static rng(*line*)

Return a rng free line

Parameters **line** – Line of logs

Returns LineColumn code, Error

Return type (str, str)

static rng_logs(*logs*)

Return a rng free line

Parameters **logs** (*str or bytes*) – Sum of logs

Returns LineColumn code, Error

Return type (str, str)

class HookTest.capitains_units.cts.**CTSMetadata_TestUnit** (**args, **kwargs*)

CTS testing object

Parameters **path** (*basestring*) – Path to the file

Variables

- **tests** – Contains the list of methods to be run again the text
- **readable** – Human friendly string associated to object methods
- **urns** – List of URN retrieved in the file.
- **type** – Type of metadata (textgroup or work)

Shared variables with parent class:

Variables

- **path** – Path for the resource
- **xml** – XML resource, parsed in python. Used to do general checking

Note: All method in `CTSText_TestUnit.tests` (“parsable”, “captain”, “metadata”, “check_urns”, “filename”) yield at least one boolean (might be more) which represents the success of it.

`captain()`

Load the file in MyCapytain

`check_urns()`

Check the validity and presence of urns in the text

Note: Populates `self.urns`

`filename()`

Check the filename and the path correctly represent the path

`metadata()`

Check the presence of all metadata

`test()`

Test a file with various checks

Returns List of urns

Return type list.<str>

class `HookTest.captains_units.cts.CTSText_TestUnit` (*path*, *countwords=False*, *time-out=30*, **args*, ***kwargs*)

CTS testing object

Parameters

- **path** (*basestring*) – Path to the file
- **countwords** (*bool*) – Count the number of words and log it if necessary

Variables

- **tests** – Contains the list of methods to be run again the text
- **readable** – Human friendly string associated to object methods
- **inv** – List of URN retrieved in metadata. Used to check the availability of metadata for the text
- **scheme** – Scheme to be used to check the
- **Text** – Text object according to MyCapytains parsing. Used to find passages

Shared variables with parent class:

Variables

- **path** – Path for the resource
- **xml** – XML resource, parsed in python. Used to do general checking

Note: All method in CTSText_TestUnit.tests (“parsable”, “has_urn”, “naming_convention”, “refsDecl”, “passages”, “unique_passage”, “inventory”) yield at least one boolean (might be more) which represents the success of it.

count_words ()

Count words in a file

duplicate ()

Detects duplicate references

empty ()

Detects empty references

epidoc ()

Check the original file against Epidoc rng through a java pipe

forbidden ()

Checks for forbidden characters in references

get_remote_rng (*url*)

Given a valid URL, downloads the RNG from the given URL and returns the filepath and name

Parameters **url** – the URL of the RNG

Returns filepath and name where the RNG was saved

has_urn ()

Test that a file has its urn according to CapiTainS Guidelines in its scheme

inventory ()

Check the naming convention of the file

language ()

Tests to make sure an xml:lang element is on the correct node

local_file ()

Check the original file against TEI rng through a java pipe

naming_convention ()

Check the naming convention of the file

parsable ()

Chacke that the text is parsable (as XML) and ingest it through MyCapytain then.

Note: Override super(parsable) and add CapiTainS Ingesting to it

passages ()

Check that passages are available at each level. On top of that, it checks for forbidden characters and duplicate in references

refsDecl ()

Check that the text contains refsDecl informations

run_rng (*rng_path*)

Run the RNG through JingTrang

Parameters **rng_path** – Path to the RelaxNG file to run against the XML to test

tei ()

Check the original file against TEI rng through a java pipe

test (*scheme, guidelines, rng=None, inventory=None*)

Test a file with various checks

Parameters

- **scheme** (*str*) – Test with TEI DTD
- **inventory** (*list*) – URNs to be matched against

Returns Iterator containing human readable test name, boolean status and logs

Return type iterator(str, bool, list(str))

unique_passage ()

Check that citation scheme do not collide (eg. Where text:1 would be the same node as text:1.1)

C

captain() (HookTest.capitains_units.cts.CTSMetadata_TestUnit method), 16
 check_urns() (HookTest.capitains_units.cts.CTSMetadata_TestUnit method), 16
 cmd() (in module HookTest.cmd), 11
 cmd() (in module HookTest.test), 14
 count_words() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17
 cover() (HookTest.test.Test method), 12
 create_manifest() (HookTest.test.Test method), 13
 CTSMetadata_TestUnit (class in HookTest.capitains_units.cts), 15
 CTSText_TestUnit (class in HookTest.capitains_units.cts), 16

D

DefaultFinder (class in HookTest.test), 12
 dict (HookTest.test.UnitLog attribute), 15
 directory (HookTest.test.Test attribute), 13
 download() (HookTest.test.Test method), 13
 duplicate() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

E

empty() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17
 end() (HookTest.test.Test method), 13
 epidoc() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

F

filename() (HookTest.capitains_units.cts.CTSMetadata_TestUnit method), 16
 FilterFinder (class in HookTest.test), 12
 find() (HookTest.test.DefaultFinder method), 12
 find() (HookTest.test.FilterFinder method), 12
 find() (HookTest.test.Test method), 13
 flush() (HookTest.test.Test method), 13

forbidden() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

G

get_remote_rng() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

H

has_urn() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

I

inventory() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

J

json (HookTest.test.Test attribute), 13

L

language() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17
 local_file() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17
 log() (HookTest.test.Test method), 13

M

metadata() (HookTest.capitains_units.cts.CTSMetadata_TestUnit method), 16
 middle() (HookTest.test.Test method), 13

N

naming_convention() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

P

parsable() (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

`parsable()` (HookTest.units.TESTUnit method), 15
`parse_args()` (in module HookTest.cmd), 11
`passages()` (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

R

`refsDecl()` (HookTest.capitains_units.cts.CTSText_TestUnit method), 17
`report` (HookTest.test.Test attribute), 13
`rng()` (HookTest.units.TESTUnit static method), 15
`rng_logs()` (HookTest.units.TESTUnit static method), 15
`run()` (HookTest.test.Test method), 13
`run_rng()` (HookTest.capitains_units.cts.CTSText_TestUnit method), 17

S

`send()` (HookTest.test.Test method), 14
`send_to_hook_from_travis()` (HookTest.test.Test method), 14
`stack` (HookTest.test.Test attribute), 14
`start()` (HookTest.test.Test method), 14
`status` (HookTest.test.Test attribute), 14
`successes` (HookTest.test.Test attribute), 14

T

`tei()` (HookTest.capitains_units.cts.CTSText_TestUnit method), 18
`Test` (class in HookTest.test), 12
`test()` (HookTest.capitains_units.cts.CTSMetadata_TestUnit method), 16
`test()` (HookTest.capitains_units.cts.CTSText_TestUnit method), 18
`TESTUnit` (class in HookTest.units), 15
`triggering_size` (HookTest.test.Test attribute), 14

U

`unique_passage()` (HookTest.capitains_units.cts.CTSText_TestUnit method), 18
`unit()` (HookTest.test.Test method), 14
`UnitLog` (class in HookTest.test), 15