inscriptis Documentation

Release 2.5.0

Albert Weichselbraun and Fabian Odoni

Mar 05, 2024

CONTENTS

A python based HTML to text conversion library, command line client and Web service with support for **nested tables**, a **subset of CSS** and optional support for providing an **annotated output**.

Inscriptis is particularly well suited for applications that require high-performance, high-quality (i.e., layout-aware) text representations of HTML content, and will aid knowledge extraction and data science tasks conducted upon Web data.

Please take a look at the Rendering document for a demonstration of inscriptis' conversion quality.

A Java port of inscriptis 1.x has been published by x28.

This document provides a short introduction to Inscriptis.

- The full documentation is built automatically and published on Read the Docs.
- If you are interested in a more general overview on the topic of *text extraction from HTML*, this blog post on different HTML to text conversion approaches, and criteria for selecting them might be interesting to you.

Table of contents

CHAPTER

ONE

STATEMENT OF NEED - WHY INSCRIPTIS?

1. Inscriptis provides a **layout-aware** conversion of HTML that more closely resembles the rendering obtained from standard Web browsers and, therefore, better preserves the spatial arrangement of text elements.

   Conversion quality becomes a factor once you need to move beyond simple HTML snippets. Non-specialized approaches and less sophisticated libraries do not correctly interpret HTML semantics and, therefore, fail to properly convert constructs such as itemizations, enumerations, and tables.

   Beautiful Soup's `get_text()` function, for example, converts the following HTML enumeration to the string `firstsecond`.

   ```
   <ul>
     <li>first</li>
     <li>second</li>
   <ul>
   ```

   Inscriptis, in contrast, not only returns the correct output

   ```
   * first
   * second
   ```

   but also supports much more complex constructs such as nested tables and also interprets a subset of HTML (e.g., `align`, `valign`) and CSS (e.g., `display`, `white-space`, `margin-top`, `vertical-align`, etc.) attributes that determine the text alignment. Any time the spatial alignment of text is relevant (e.g., for many knowledge extraction tasks, the computation of word embeddings and language models, and sentiment analysis) an accurate HTML to text conversion is essential.

2. Inscriptis supports *annotation rules*, i.e., user-provided mappings that allow for annotating the extracted text based on structural and semantic information encoded in HTML tags and attributes used for controlling structure and layout in the original HTML document. These rules might be used to

   - provide downstream knowledge extraction components with additional information that may be leveraged to improve their respective performance.
   - assist manual document annotation processes (e.g., for qualitative analysis or gold standard creation). `Inscriptis` supports multiple export formats such as XML, annotated HTML and the JSONL format that is used by the open source annotation tool doccano.
   - enabling the use of `Inscriptis` for tasks such as content extraction (i.e., extract task-specific relevant content from a Web page) which rely on information on the HTML document's structure.

CHAPTER

TWO

INSTALLATION

At the command line:

```
$ pip install inscriptis
```

Or, if you don't have pip installed:

```
$ easy_install inscriptis
```

CHAPTER

THREE

PYTHON LIBRARY

Embedding inscriptis into your code is easy, as outlined below:

```
import urllib.request
from inscriptis import get_text

url = "https://www.fhgr.ch"
html = urllib.request.urlopen(url).read().decode('utf-8')

text = get_text(html)
print(text)
```

CHAPTER

FOUR

STANDALONE COMMAND LINE CLIENT

The command line client converts HTML files or text retrieved from Web pages to the corresponding text representation.

4.1 Command line parameters

The inscript command line client supports the following parameters:

```
usage: inscript [-h] [-o OUTPUT] [-e ENCODING] [-i] [-d] [-l] [-a] [-r ANNOTATION_RULES]␣
↪[-p POSTPROCESSOR] [--indentation INDENTATION]
                  [--table-cell-separator TABLE_CELL_SEPARATOR] [-v]
                  [input]

Convert the given HTML document to text.

positional arguments:
  input                 Html input either from a file or a URL (default:stdin).

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Output file (default:stdout).
  -e ENCODING, --encoding ENCODING
                        Input encoding to use (default:utf-8 for files; detected server␣
↪encoding for Web URLs).
  -i, --display-image-captions
                        Display image captions (default:false).
  -d, --deduplicate-image-captions
                        Deduplicate image captions (default:false).
  -l, --display-link-targets
                        Display link targets (default:false).
  -a, --display-anchor-urls
                        Display anchor URLs (default:false).
  -r ANNOTATION_RULES, --annotation-rules ANNOTATION_RULES
                        Path to an optional JSON file containing rules for annotating␣
↪the retrieved text.
  -p POSTPROCESSOR, --postprocessor POSTPROCESSOR
                        Optional component for postprocessing the result (html, surface,␣
↪xml).
  --indentation INDENTATION
                        How to handle indentation (extended or strict; default:␣
```

(continues on next page)

(continued from previous page)

```
↪extended).
  --table-cell-separator TABLE_CELL_SEPARATOR
                        Separator to use between table cells (default: three spaces).
  -v, --version         display version information
```

4.2 HTML to text conversion

convert the given page to text and output the result to the screen:

```
$ inscript https://www.fhgr.ch
```

convert the file to text and save the output to fhgr.txt:

```
$ inscript fhgr.html -o fhgr.txt
```

convert the file using strict indentation (i.e., minimize indentation and extra spaces) and save the output to fhgr-layout-optimized.txt:

```
$ inscript --indentation strict fhgr.html -o fhgr-layout-optimized.txt
```

convert HTML provided via stdin and save the output to output.txt:

```
$ echo "<body><p>Make it so!</p></body>" | inscript -o output.txt
```

4.3 HTML to annotated text conversion

convert and annotate HTML from a Web page using the provided annotation rules.

Download the example annotation-profile.json and save it to your working directory:

```
$ inscript https://www.fhgr.ch -r annotation-profile.json
```

The annotation rules are specified in *annotation-profile.json*:

```
{
 "h1": ["heading", "h1"],
 "h2": ["heading", "h2"],
 "b": ["emphasis"],
 "div#class=toc": ["table-of-contents"],
 "#class=FactBox": ["fact-box"],
 "#cite": ["citation"]
}
```

The dictionary maps an HTML tag and/or attribute to the annotations inscriptis should provide for them. In the example above, for instance, the tag `h1` yields the annotations `heading` and `h1`, a `div` tag with a `class` that contains the value `toc` results in the annotation `table-of-contents`, and all tags with a `cite` attribute are annotated with `citation`.

Given these annotation rules the HTML file

```
<h1>Chur</h1>
<b>Chur</b> is the capital and largest town of the Swiss canton of the
Grisons and lies in the Grisonian Rhine Valley.
```

yields the following JSONL output

```
{"text": "Chur\n\nChur is the capital and largest town of the Swiss canton
          of the Grisons and lies in the Grisonian Rhine Valley.",
 "label": [[0, 4, "heading"], [0, 4, "h1"], [6, 10, "emphasis"]]}
```

The provided list of labels contains all annotated text elements with their start index, end index and the assigned label.

4.4 Annotation postprocessors

Annotation postprocessors enable the post processing of annotations to formats that are suitable for your particular application. Post processors can be specified with the `-p` or `--postprocessor` command line argument:

```
$ inscript https://www.fhgr.ch \
        -r ./annotation/examples/annotation-profile.json \
        -p surface
```

Output:

```
{"text": "  Chur\n\n  Chur is the capital and largest town of the Swiss
          canton of the Grisons and lies in the Grisonian Rhine Valley.",
 "label": [[0, 6, "heading"], [8, 14, "emphasis"]],
 "tag": "<heading>Chur</heading>\n\n<emphasis>Chur</emphasis> is the
        capital and largest town of the Swiss canton of the Grisons and
        lies in the Grisonian Rhine Valley."}
```

Currently, inscriptis supports the following postprocessors:

- surface: returns a list of mapping between the annotation's surface form and its label:

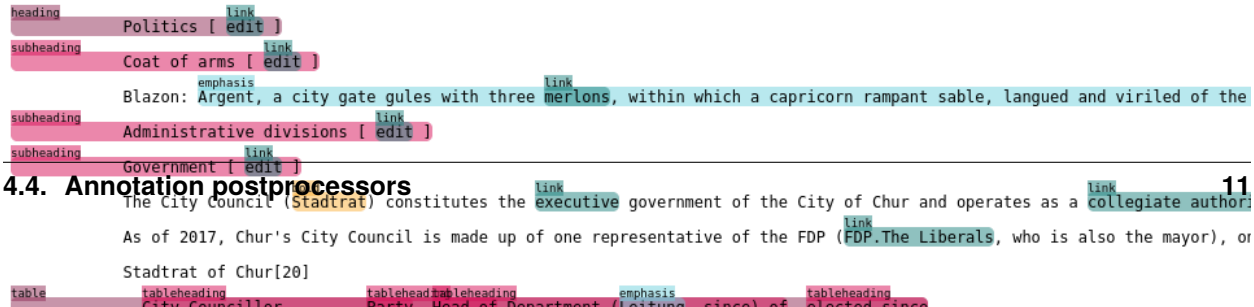
  ```
  [
     ['heading', 'Chur'],
     ['emphasis': 'Chur']
  ]
  ```

- xml: returns an additional annotated text version:

  ```
  <?xml version="1.0" encoding="UTF-8" ?>
  <heading>Chur</heading>

  <emphasis>Chur</emphasis> is the capital and largest town of the Swiss
  canton of the Grisons and lies in the Grisonian Rhine Valley.
  ```

- html: creates an HTML file which contains the converted text and highlights all annotations as outlined below:



CHAPTER

FIVE

WEB SERVICE

A FastAPI-based Web Service that uses Inscriptis for translating HTML pages to plain text.

5.1 Run the Web Service on your host system

Install the optional feature *web-service* for inscriptis:

```
$ pip install inscriptis[web-service]
```

Start the Inscriptis Web service with the following command:

```
$ uvicorn inscriptis.service.web:app --port 5000 --host 127.0.0.1
```

5.2 Run the Web Service with Docker

The docker definition can be found here:

```
$ docker pull ghcr.io/weblyzard/inscriptis:latest
$ docker run -n inscriptis ghcr.io/weblyzard/inscriptis:latest
```

5.3 Run as Kubernetes Deployment

The helm chart for deployment on a kubernetes cluster is located in the inscriptis-helm repository.

5.4 Use the Web Service

The Web services receives the HTML file in the request body and returns the corresponding text. The file's encoding needs to be specified in the `Content-Type` header (`UTF-8` in the example below):

```
$ curl -X POST  -H "Content-Type: text/html; encoding=UTF8"  \
       --data-binary @test.html  http://localhost:5000/get_text
```

The service also supports a version call:

```
$ curl http://localhost:5000/version
```

CHAPTER

SIX

EXAMPLE ANNOTATION PROFILES

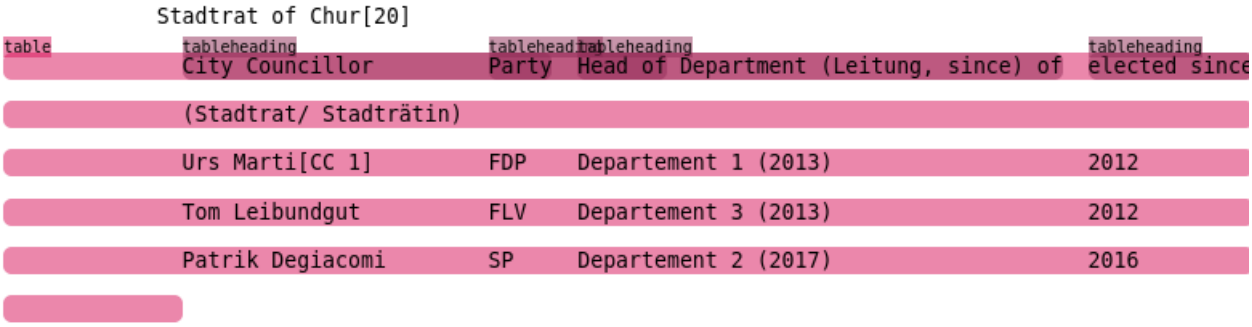
The following section provides a number of example annotation profiles illustrating the use of Inscriptis' annotation support. The examples present the used annotation rules and an image that highlights a snippet with the annotated text on the converted web page, which has been created using the HTML postprocessor as outlined in Section *annotation postprocessors*.

6.1 Wikipedia tables and table metadata

The following annotation rules extract tables from Wikipedia pages, and annotate table headings that are typically used to indicate column or row headings.

```
{
   "table": ["table"],
   "th": ["tableheading"],
   "caption": ["caption"]
}
```

The figure below outlines an example table from Wikipedia that has been annotated using these rules.

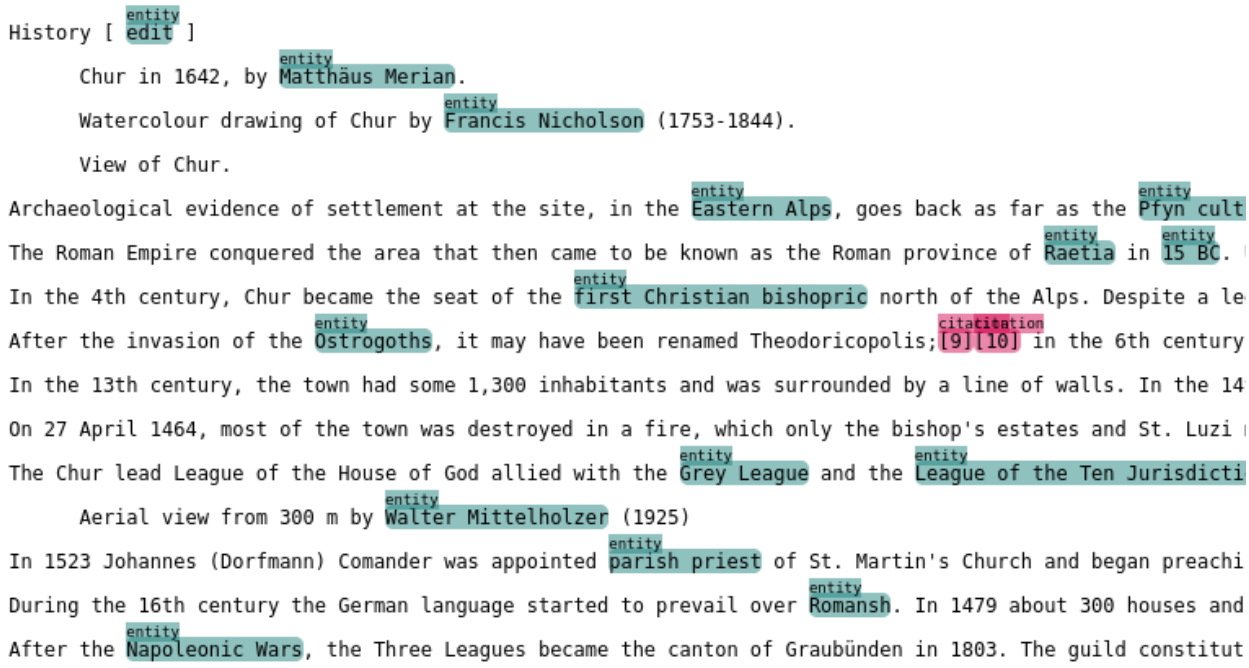


6.2 References to entities, missing entities and citations from Wikipedia

This profile extracts references to Wikipedia entities, missing entities and citations. Please note that the profile isn't perfect, since it also annotates `[ edit ]` links.

```
{
   "a#title": ["entity"],
   "a#class=new": ["missing"],
   "class=reference": ["citation"]
}
```

The figure shows entities and citations that have been identified on a Wikipedia page using these rules.



6.3 Posts and post metadata from the XDA developer forum

The annotation rules below, extract posts with metadata on the post's time, user and the user's job title from the XDA developer forum.

```
{
    "article#class=message-body": ["article"],
    "li#class=u-concealed": ["time"],
    "#itemprop=name": ["user-name"],
    "#itemprop=jobTitle": ["user-title"]
}
```

The figure illustrates the annotated metadata on posts from the XDA developer forum.

time
* Jan 3, 2021 at 11:13 PM
*
* #3
article
I keep seeing no SD card slot in none of the S21's. I can't believe it!
user-name
blackhawk
user-title
Senior Member
Jun 23, 2020 4,746 1,294
time
* Jan 3, 2021 at 11:53 PM
*
* #4
article
tailgunner9 said:
NO S21's for me. They do Not have SD Card slot !!!!!!
Click to expand...
Click to collapse
No SD card slot, no sale.
I want my bloody data drive... no compromise on that.
*
Reactions: Chef_of_Sweden
user-name
Geekser
user-title
Senior Member

6.4 Code and metadata from Stackoverflow pages

The rules below extracts code and metadata on users and comments from Stackoverflow pages.

```
{
   "code": ["code"],
   "#itemprop=dateCreated": ["creation-date"],
   "#class=user-details": ["user"],
   "#class=reputation-score": ["reputation"],
   "#class=comment-date": ["comment-date"],
   "#class=comment-copy": ["comment-comment"]
}
```

Applying these rules to a Stackoverflow page on text extraction from HTML yields the following snippet:

```
code
                    from htmllib import HTMLParser, HTMLParseError

                    from formatter import AbstractFormatter, DumbWriter

                    p = HTMLParser(AbstractFormatter(DumbWriter()))

                    try: p.feed('hello
there'); p.close() #calling close is not usually needed, but let's play it safe

                    except HTMLParseError: print ':(' #the html is badly malformed (or you found a bug)

                                  Share

                                  Improve this answer

                                  Follow

                                edited Jul 19 '15 at 0:57
user
                                Ponkadoodle
reputation
                                  5,523 4 4 gold badges 33 33 silver badges 61 61 bronze badges

                                answered Feb 20 '12 at 6:39
user
                                Mark Mark
reputation
                                  41 1 1 bronze badge

                1
comment-comment
                            * NB: HTMLError and HTMLParserError should both read HTMLParseError. This works,

                                  – Dave Knight
comment-date
                                Apr 8 '14 at 8:09
```

CHAPTER

SEVEN

ADVANCED TOPICS

7.1 Annotated text

Inscriptis can provide annotations alongside the extracted text which allows downstream components to draw upon semantics that have only been available in the original HTML file.

The extracted text and annotations can be exported in different formats, including the popular JSONL format which is used by doccano.

Example output:

```
{"text": "Chur\n\nChur is the capital and largest town of the Swiss canton
          of the Grisons and lies in the Grisonian Rhine Valley.",
 "label": [[0, 4, "heading"], [0, 4, "h1"], [6, 10, "emphasis"]]}
```

The output above is produced, if inscriptis is run with the following annotation rules:

```
{
 "h1": ["heading", "h1"],
 "b": ["emphasis"],
}
```

The code below demonstrates how inscriptis' annotation capabilities can be used within a program:

```
import urllib.request
from inscriptis import get_annotated_text
from inscriptis.model.config import ParserConfig

url = "https://www.fhgr.ch"
html = urllib.request.urlopen(url).read().decode('utf-8')

rules = {'h1': ['heading', 'h1'],
         'h2': ['heading', 'h2'],
         'b': ['emphasis'],
         'table': ['table']
        }

output = get_annotated_text(html, ParserConfig(annotation_rules=rules)
print("Text:", output['text'])
print("Annotations:", output['label'])
```

7.2 Fine tuning

The following options are available for fine tuning inscriptis' HTML rendering:

1. **More rigorous indentation:** call `inscriptis.get_text()` with the parameter `indentation='extended'` to also use indentation for tags such as and that do not provide indentation in their standard definition. This strategy is the default in `inscript` and many other tools such as Lynx. If you do not want extended indentation you can use the parameter `indentation='standard'` instead.
2. **Overwriting the default CSS definition:** inscriptis uses CSS definitions that are maintained in `inscriptis.css.CSS` for rendering HTML tags. You can override these definitions (and therefore change the rendering) as outlined below:

```
from lxml.html import fromstring
from inscriptis.css_profiles import CSS_PROFILES, HtmlElement
from inscriptis.html_properties import Display
from inscriptis.model.config import ParserConfig

# create a custom CSS based on the default style sheet and change the
# rendering of `div` and `span` elements
css = CSS_PROFILES['strict'].copy()
css['div'] = HtmlElement(display=Display.block, padding=2)
css['span'] = HtmlElement(prefix=' ', suffix=' ')

html_tree = fromstring(html)
# create a parser using a custom css
config = ParserConfig(css=css)
parser = Inscriptis(html_tree, config)
text = parser.get_text()
```

7.3 Custom HTML tag handling

If the fine-tuning options discussed above are not sufficient, you may even override Inscriptis' handling of start and end tags as outlined below:

```
from inscriptis import ParserConfig
from inscriptis.html_engine import Inscriptis
from inscriptis.model.tag import CustomHtmlTagHandlerMapping

my_mapping = CustomHtmlTagHandlerMapping(
    start_tag_mapping={'a': my_handle_start_a},
    end_tag_mapping={'a': my_handle_end_a}
)
inscriptis = Inscriptis(html_tree,
                        ParserConfig(custom_html_tag_handler_mapping=my_mapping))
text = inscriptis.get_text()
```

In the example the standard HTML handlers for the `a` tag are overwritten with custom versions (i.e., `my_handle_start_a` and `my_handle_end_a`). You may define custom handlers for any tag, regardless of whether it already exists in the standard mapping.

Please refer to custom-html-handling.py for a working example. The standard HTML tag handlers can be found in the inscriptis.model.tag package.

7.4 Optimizing memory consumption

Inscriptis uses the Python lxml library which prefers to reuse memory rather than release it to the operating system. This behavior might lead to an increased memory consumption, if you use inscriptis within a Web service that parses very complex HTML pages.

The following code mitigates this problem on Unix systems by manually forcing lxml to release the allocated memory:

```
import ctypes
def trim_memory() -> int:
   libc = ctypes.CDLL("libc.so.6")
   return libc.malloc_trim(0)
```

CHAPTER

EIGHT

EXAMPLES

8.1 Strict indentation handling

The following example demonstrates modifying `ParserConfig` for strict indentation handling.

```
from inscriptis import get_text
from inscriptis.css_profiles import CSS_PROFILES
from inscriptis.model.config import ParserConfig

config = ParserConfig(css=CSS_PROFILES['strict'].copy())
text = get_text('fi<span>r</span>st', config)
print(text)
```

8.2 Ignore elements during parsing

Overwriting the default CSS profile also allows changing the rendering of selected elements. The snippet below, for example, removes forms from the parsed text by setting the definition of the `form` tag to `Display.none`.

```
from inscriptis import get_text
from inscriptis.css_profiles import CSS_PROFILES, HtmlElement
from inscriptis.html_properties import Display
from inscriptis.model.config import ParserConfig

# create a custom CSS based on the default style sheet and change the
# rendering of `div` and `span` elements
css = CSS_PROFILES['strict'].copy()
css['form'] = HtmlElement(display=Display.none)

# create a parser configuration using a custom css
html = """First line.
          <form>
            User data
            <label for="name">Name:</label><br>
            <input type="text" id="name" name="name"><br>
            <label for="pass">Password:</label><br>
            <input type="hidden" id="pass" name="pass">
          </form>"""
config = ParserConfig(css=css)
```

(continues on next page)

(continued from previous page)

```
text = get_text(html, config)
print(text)
```

CHAPTER

NINE

CITATION

There is a Journal of Open Source Software paper you can cite for Inscriptis:

```
@article{Weichselbraun2021,
  doi = {10.21105/joss.03557},
  url = {https://doi.org/10.21105/joss.03557},
  year = {2021},
  publisher = {The Open Journal},
  volume = {6},
  number = {66},
  pages = {3557},
  author = {Albert Weichselbraun},
  title = {Inscriptis - A Python-based HTML to text conversion library optimized for␣
↪knowledge extraction from the Web},
  journal = {Journal of Open Source Software}
}
```

CHAPTER

TEN

CHANGELOG

A full list of changes can be found in the release notes.

CHAPTER

ELEVEN

DOCUMENTATION

Contents:

11.1 inscriptis – HTML to text conversion library, command line client and Web service

A python based HTML to text conversion library, command line client and Web service with support for **nested tables**, a **subset of CSS** and optional support for providing an **annotated output**.

Inscriptis is particularly well suited for applications that require high-performance, high-quality (i.e., layout-aware) text representations of HTML content, and will aid knowledge extraction and data science tasks conducted upon Web data.

Please take a look at the Rendering document for a demonstration of inscriptis' conversion quality.

A Java port of inscriptis 1.x has been published by x28.

This document provides a short introduction to Inscriptis.

- The full documentation is built automatically and published on Read the Docs.
- If you are interested in a more general overview on the topic of *text extraction from HTML*, this blog post on different HTML to text conversion approaches, and criteria for selecting them might be interesting to you.

Table of contents

11.1.1 Statement of need - why inscriptis?

1. Inscriptis provides a **layout-aware** conversion of HTML that more closely resembles the rendering obtained from standard Web browsers and, therefore, better preserves the spatial arrangement of text elements.

   Conversion quality becomes a factor once you need to move beyond simple HTML snippets. Non-specialized approaches and less sophisticated libraries do not correctly interpret HTML semantics and, therefore, fail to properly convert constructs such as itemizations, enumerations, and tables.

   Beautiful Soup's `get_text()` function, for example, converts the following HTML enumeration to the string `firstsecond`.

```
<ul>
  <li>first</li>
  <li>second</li>
<ul>
```

Inscriptis, in contrast, not only returns the correct output

```
* first
* second
```

but also supports much more complex constructs such as nested tables and also interprets a subset of HTML (e.g., `align`, `valign`) and CSS (e.g., `display`, `white-space`, `margin-top`, `vertical-align`, etc.) attributes that determine the text alignment. Any time the spatial alignment of text is relevant (e.g., for many knowledge extraction tasks, the computation of word embeddings and language models, and sentiment analysis) an accurate HTML to text conversion is essential.

2. Inscriptis supports *annotation rules*, i.e., user-provided mappings that allow for annotating the extracted text based on structural and semantic information encoded in HTML tags and attributes used for controlling structure and layout in the original HTML document. These rules might be used to

   - provide downstream knowledge extraction components with additional information that may be leveraged to improve their respective performance.
   - assist manual document annotation processes (e.g., for qualitative analysis or gold standard creation). `Inscriptis` supports multiple export formats such as XML, annotated HTML and the JSONL format that is used by the open source annotation tool doccano.
   - enabling the use of `Inscriptis` for tasks such as content extraction (i.e., extract task-specific relevant content from a Web page) which rely on information on the HTML document's structure.

11.1.2 Installation

At the command line:

```
$ pip install inscriptis
```

Or, if you don't have pip installed:

```
$ easy_install inscriptis
```

11.1.3 Python library

Embedding inscriptis into your code is easy, as outlined below:

```
import urllib.request
from inscriptis import get_text

url = "https://www.fhgr.ch"
html = urllib.request.urlopen(url).read().decode('utf-8')

text = get_text(html)
print(text)
```

11.1.4 Standalone command line client

The command line client converts HTML files or text retrieved from Web pages to the corresponding text representation.

Command line parameters

The inscript command line client supports the following parameters:

```
usage: inscript [-h] [-o OUTPUT] [-e ENCODING] [-i] [-d] [-l] [-a] [-r ANNOTATION_RULES]␣
↪[-p POSTPROCESSOR] [--indentation INDENTATION]
                    [--table-cell-separator TABLE_CELL_SEPARATOR] [-v]
                    [input]

Convert the given HTML document to text.

positional arguments:
  input                 Html input either from a file or a URL (default:stdin).

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Output file (default:stdout).
  -e ENCODING, --encoding ENCODING
                        Input encoding to use (default:utf-8 for files; detected server␣
↪encoding for Web URLs).
  -i, --display-image-captions
                        Display image captions (default:false).
  -d, --deduplicate-image-captions
                        Deduplicate image captions (default:false).
  -l, --display-link-targets
                        Display link targets (default:false).
  -a, --display-anchor-urls
                        Display anchor URLs (default:false).
  -r ANNOTATION_RULES, --annotation-rules ANNOTATION_RULES
                        Path to an optional JSON file containing rules for annotating␣
↪the retrieved text.
  -p POSTPROCESSOR, --postprocessor POSTPROCESSOR
                        Optional component for postprocessing the result (html, surface,␣
↪xml).
  --indentation INDENTATION
                        How to handle indentation (extended or strict; default:␣
↪extended).
  --table-cell-separator TABLE_CELL_SEPARATOR
                        Separator to use between table cells (default: three spaces).
  -v, --version         display version information
```

HTML to text conversion

convert the given page to text and output the result to the screen:

```
$ inscript https://www.fhgr.ch
```

convert the file to text and save the output to fhgr.txt:

```
$ inscript fhgr.html -o fhgr.txt
```

convert the file using strict indentation (i.e., minimize indentation and extra spaces) and save the output to fhgr-layout-optimized.txt:

```
$ inscript --indentation strict fhgr.html -o fhgr-layout-optimized.txt
```

convert HTML provided via stdin and save the output to output.txt:

```
$ echo "<body><p>Make it so!</p></body>" | inscript -o output.txt
```

HTML to annotated text conversion

convert and annotate HTML from a Web page using the provided annotation rules.

Download the example annotation-profile.json and save it to your working directory:

```
$ inscript https://www.fhgr.ch -r annotation-profile.json
```

The annotation rules are specified in *annotation-profile.json*:

```
{
 "h1": ["heading", "h1"],
 "h2": ["heading", "h2"],
 "b": ["emphasis"],
 "div#class=toc": ["table-of-contents"],
 "#class=FactBox": ["fact-box"],
 "#cite": ["citation"]
}
```

The dictionary maps an HTML tag and/or attribute to the annotations inscriptis should provide for them. In the example above, for instance, the tag `h1` yields the annotations `heading` and `h1`, a `div` tag with a `class` that contains the value `toc` results in the annotation `table-of-contents`, and all tags with a `cite` attribute are annotated with `citation`.

Given these annotation rules the HTML file

```
<h1>Chur</h1>
<b>Chur</b> is the capital and largest town of the Swiss canton of the
Grisons and lies in the Grisonian Rhine Valley.
```

yields the following JSONL output

```
{"text": "Chur\n\nChur is the capital and largest town of the Swiss canton
          of the Grisons and lies in the Grisonian Rhine Valley.",
 "label": [[0, 4, "heading"], [0, 4, "h1"], [6, 10, "emphasis"]]}
```

The provided list of labels contains all annotated text elements with their start index, end index and the assigned label.

Annotation postprocessors

Annotation postprocessors enable the post processing of annotations to formats that are suitable for your particular application. Post processors can be specified with the `-p` or `--postprocessor` command line argument:

```
$ inscript https://www.fhgr.ch \
        -r ./annotation/examples/annotation-profile.json \
        -p surface
```

Output:

```
{"text": "  Chur\n\n  Chur is the capital and largest town of the Swiss
          canton of the Grisons and lies in the Grisonian Rhine Valley.",
 "label": [[0, 6, "heading"], [8, 14, "emphasis"]],
 "tag": "<heading>Chur</heading>\n\n<emphasis>Chur</emphasis> is the
        capital and largest town of the Swiss canton of the Grisons and
        lies in the Grisonian Rhine Valley."}
```

Currently, inscriptis supports the following postprocessors:

- surface: returns a list of mapping between the annotation's surface form and its label:

  ```
  [
     ['heading', 'Chur'],
     ['emphasis': 'Chur']
  ]
  ```

- xml: returns an additional annotated text version:

  ```
  <?xml version="1.0" encoding="UTF-8" ?>
  <heading>Chur</heading>

  <emphasis>Chur</emphasis> is the capital and largest town of the Swiss
  canton of the Grisons and lies in the Grisonian Rhine Valley.
  ```

- html: creates an HTML file which contains the converted text and highlights all annotations as outlined below:

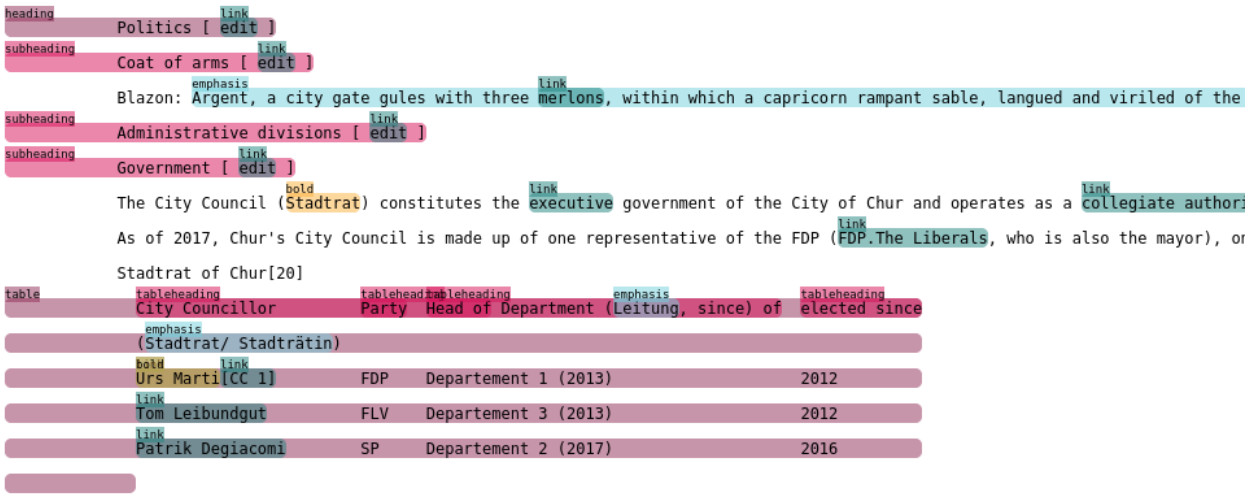


Fig. 1: Snippet of the rendered HTML file created with the following command line options and annotation rules:

```
inscript --annotation-rules ./wikipedia.json \
        --postprocessor html \
        https://en.wikipedia.org/wiki/Chur.html
```

Annotation rules encoded in the `wikipedia.json` file:

11.1.5 Web Service

A FastAPI-based Web Service that uses Inscriptis for

translating HTML pages to plain text.

Run the Web Service on your host system

Install the optional feature *web-service* for inscriptis:

```
$ pip install inscriptis[web-service]
```

Start the Inscriptis Web service with the following command:

```
$ uvicorn inscriptis.service.web:app --port 5000 --host 127.0.0.1
```

Run the Web Service with Docker

The docker definition can be found here:

```
$ docker pull ghcr.io/weblyzard/inscriptis:latest
$ docker run -n inscriptis ghcr.io/weblyzard/inscriptis:latest
```

Run as Kubernetes Deployment

The helm chart for deployment on a kubernetes cluster is located in the inscriptis-helm repository.

Use the Web Service

The Web services receives the HTML file in the request body and returns the corresponding text. The file's encoding needs to be specified in the `Content-Type` header (`UTF-8` in the example below):

```
$ curl -X POST  -H "Content-Type: text/html; encoding=UTF8"  \
       --data-binary @test.html  http://localhost:5000/get_text
```

The service also supports a version call:

```
$ curl http://localhost:5000/version
```

11.1.6 Example annotation profiles

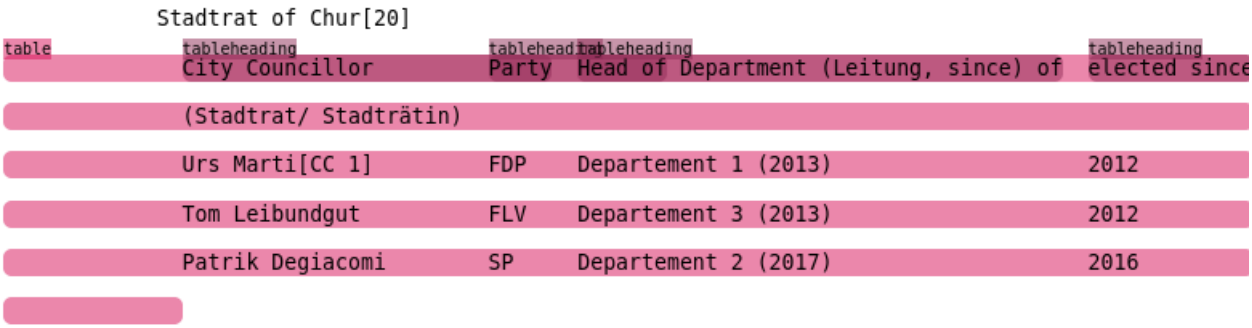
The following section provides a number of example annotation profiles illustrating the use of Inscriptis' annotation support. The examples present the used annotation rules and an image that highlights a snippet with the annotated text on the converted web page, which has been created using the HTML postprocessor as outlined in Section *annotation postprocessors*.

Wikipedia tables and table metadata

The following annotation rules extract tables from Wikipedia pages, and annotate table headings that are typically used to indicate column or row headings.

```
{
   "table": ["table"],
   "th": ["tableheading"],
   "caption": ["caption"]
}
```

The figure below outlines an example table from Wikipedia that has been annotated using these rules.

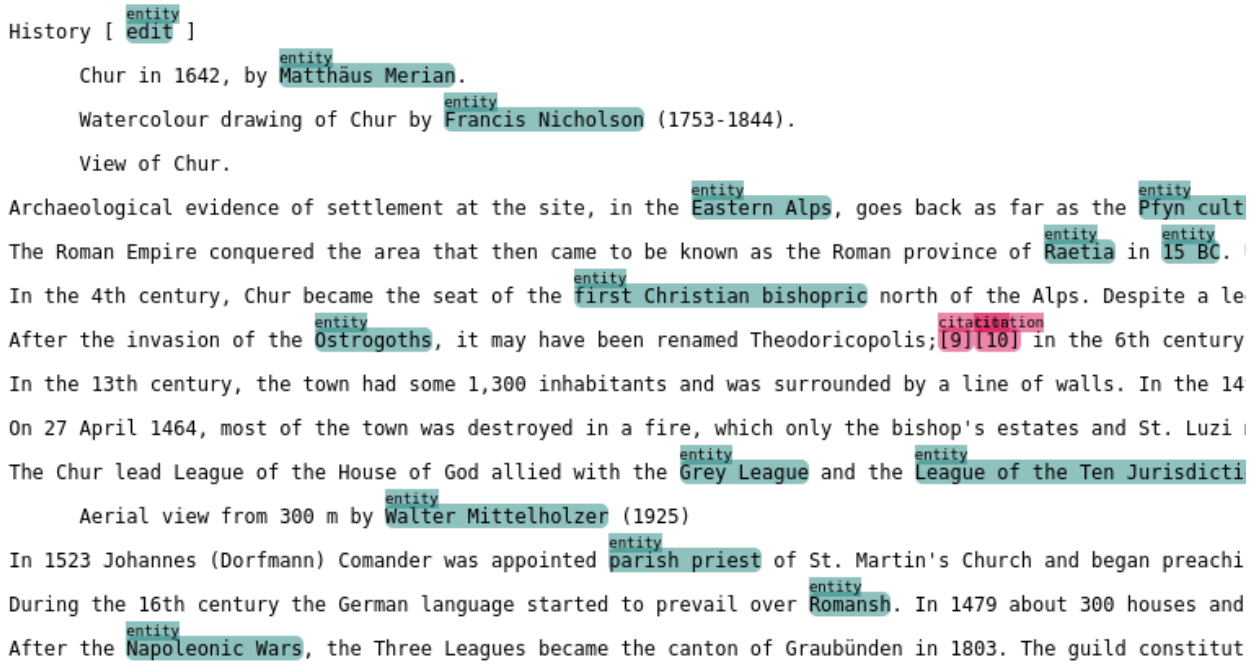


References to entities, missing entities and citations from Wikipedia

This profile extracts references to Wikipedia entities, missing entities and citations. Please note that the profile isn't perfect, since it also annotates `[ edit ]` links.

```
{
   "a#title": ["entity"],
   "a#class=new": ["missing"],
   "class=reference": ["citation"]
}
```

The figure shows entities and citations that have been identified on a Wikipedia page using these rules.



Posts and post metadata from the XDA developer forum

The annotation rules below, extract posts with metadata on the post's time, user and the user's job title from the XDA developer forum.

```
{
    "article#class=message-body": ["article"],
    "li#class=u-concealed": ["time"],
    "#itemprop=name": ["user-name"],
    "#itemprop=jobTitle": ["user-title"]
}
```

The figure illustrates the annotated metadata on posts from the XDA developer forum.

```
time
                                * Jan 3, 2021 at 11:13 PM
                                *
                                * #3
article
                                    I keep seeing no SD card slot in none of the S21's. I can't believe it!
user-name
                                blackhawk
user-title
                                Senior Member
                                  Jun 23, 2020 4,746 1,294
time
                                * Jan 3, 2021 at 11:53 PM
                                *
                                * #4
article
                                      tailgunner9 said:
                                       NO S21's for me. They do Not have SD Card slot !!!!!!
                                       Click to expand...
                                       Click to collapse
                                    No SD card slot, no sale.
                                    I want my bloody data drive... no compromise on that.
                                  *
                                Reactions: Chef_of_Sweden
user-name
                                Geekser
user-title
                                Senior Member
```

Code and metadata from Stackoverflow pages

The rules below extracts code and metadata on users and comments from Stackoverflow pages.

```
{
   "code": ["code"],
   "#itemprop=dateCreated": ["creation-date"],
   "#class=user-details": ["user"],
   "#class=reputation-score": ["reputation"],
   "#class=comment-date": ["comment-date"],
   "#class=comment-copy": ["comment-comment"]
}
```

Applying these rules to a Stackoverflow page on text extraction from HTML yields the following snippet:

```
code
                    from htmllib import HTMLParser, HTMLParseError

                    from formatter import AbstractFormatter, DumbWriter

                    p = HTMLParser(AbstractFormatter(DumbWriter()))

                    try: p.feed('hello
there'); p.close() #calling close is not usually needed, but let's play it safe

                    except HTMLParseError: print ':(' #the html is badly malformed (or you found a bug)

                                  Share

                                  Improve this answer

                                  Follow

                                edited Jul 19 '15 at 0:57
user
                                Ponkadoodle
reputation
                                  5,523 4 4 gold badges 33 33 silver badges 61 61 bronze badges

                                answered Feb 20 '12 at 6:39
user
                                Mark Mark
reputation
                                  41 1 1 bronze badge

                1
comment-comment
                            * NB: HTMLError and HTMLParserError should both read HTMLParseError. This works,

                                  – Dave Knight
comment-date
                                Apr 8 '14 at 8:09
```

11.1.7 Advanced topics

Annotated text

Inscriptis can provide annotations alongside the extracted text which allows downstream components to draw upon semantics that have only been available in the original HTML file.

The extracted text and annotations can be exported in different formats, including the popular JSONL format which is used by doccano.

Example output:

```
{"text": "Chur\n\nChur is the capital and largest town of the Swiss canton
          of the Grisons and lies in the Grisonian Rhine Valley.",
 "label": [[0, 4, "heading"], [0, 4, "h1"], [6, 10, "emphasis"]]}
```

The output above is produced, if inscriptis is run with the following annotation rules:

```
{
 "h1": ["heading", "h1"],
 "b": ["emphasis"],
}
```

The code below demonstrates how inscriptis' annotation capabilities can be used within a program:

```
import urllib.request
from inscriptis import get_annotated_text
from inscriptis.model.config import ParserConfig

url = "https://www.fhgr.ch"
html = urllib.request.urlopen(url).read().decode('utf-8')

rules = {'h1': ['heading', 'h1'],
         'h2': ['heading', 'h2'],
         'b': ['emphasis'],
         'table': ['table']
        }

output = get_annotated_text(html, ParserConfig(annotation_rules=rules)
print("Text:", output['text'])
print("Annotations:", output['label'])
```

Fine tuning

The following options are available for fine tuning inscriptis' HTML rendering:

1. **More rigorous indentation:** call `inscriptis.get_text()` with the parameter `indentation='extended'` to also use indentation for tags such as and that do not provide indentation in their standard definition. This strategy is the default in `inscript` and many other tools such as Lynx. If you do not want extended indentation you can use the parameter `indentation='standard'` instead.
2. **Overwriting the default CSS definition:** inscriptis uses CSS definitions that are maintained in `inscriptis.css.CSS` for rendering HTML tags. You can override these definitions (and therefore change the rendering) as outlined below:

```
from lxml.html import fromstring
from inscriptis.css_profiles import CSS_PROFILES, HtmlElement
from inscriptis.html_properties import Display
from inscriptis.model.config import ParserConfig

# create a custom CSS based on the default style sheet and change the
# rendering of `div` and `span` elements
css = CSS_PROFILES['strict'].copy()
css['div'] = HtmlElement(display=Display.block, padding=2)
css['span'] = HtmlElement(prefix=' ', suffix=' ')
```

(continues on next page)

(continued from previous page)

```
html_tree = fromstring(html)
# create a parser using a custom css
config = ParserConfig(css=css)
parser = Inscriptis(html_tree, config)
text = parser.get_text()
```

Custom HTML tag handling

If the fine-tuning options discussed above are not sufficient, you may even override Inscriptis' handling of start and end tags as outlined below:

```
from inscriptis import ParserConfig
from inscriptis.html_engine import Inscriptis
from inscriptis.model.tag import CustomHtmlTagHandlerMapping

my_mapping = CustomHtmlTagHandlerMapping(
    start_tag_mapping={'a': my_handle_start_a},
    end_tag_mapping={'a': my_handle_end_a}
)
inscriptis = Inscriptis(html_tree,
                        ParserConfig(custom_html_tag_handler_mapping=my_mapping))
text = inscriptis.get_text()
```

In the example the standard HTML handlers for the `a` tag are overwritten with custom versions (i.e., `my_handle_start_a` and `my_handle_end_a`). You may define custom handlers for any tag, regardless of whether it already exists in the standard mapping.

Please refer to custom-html-handling.py for a working example. The standard HTML tag handlers can be found in the inscriptis.model.tag package.

Optimizing memory consumption

Inscriptis uses the Python lxml library which prefers to reuse memory rather than release it to the operating system. This behavior might lead to an increased memory consumption, if you use inscriptis within a Web service that parses very complex HTML pages.

The following code mitigates this problem on Unix systems by manually forcing lxml to release the allocated memory:

```
import ctypes
def trim_memory() -> int:
   libc = ctypes.CDLL("libc.so.6")
   return libc.malloc_trim(0)
```

11.1.8 Examples

Strict indentation handling

The following example demonstrates modifying `ParserConfig` for strict indentation handling.

```
from inscriptis import get_text
from inscriptis.css_profiles import CSS_PROFILES
from inscriptis.model.config import ParserConfig

config = ParserConfig(css=CSS_PROFILES['strict'].copy())
text = get_text('fi<span>r</span>st', config)
print(text)
```

Ignore elements during parsing

Overwriting the default CSS profile also allows changing the rendering of selected elements. The snippet below, for example, removes forms from the parsed text by setting the definition of the `form` tag to `Display.none`.

```
from inscriptis import get_text
from inscriptis.css_profiles import CSS_PROFILES, HtmlElement
from inscriptis.html_properties import Display
from inscriptis.model.config import ParserConfig

# create a custom CSS based on the default style sheet and change the
# rendering of `div` and `span` elements
css = CSS_PROFILES['strict'].copy()
css['form'] = HtmlElement(display=Display.none)

# create a parser configuration using a custom css
html = """First line.
          <form>
            User data
            <label for="name">Name:</label><br>
            <input type="text" id="name" name="name"><br>
            <label for="pass">Password:</label><br>
            <input type="hidden" id="pass" name="pass">
          </form>"""
config = ParserConfig(css=css)
text = get_text(html, config)
print(text)
```

11.1.9 Citation

There is a Journal of Open Source Software paper you can cite for Inscriptis:

```
@article{Weichselbraun2021,
  doi = {10.21105/joss.03557},
  url = {https://doi.org/10.21105/joss.03557},
  year = {2021},
  publisher = {The Open Journal},
```

(continues on next page)

(continued from previous page)

```
  volume = {6},
  number = {66},
  pages = {3557},
  author = {Albert Weichselbraun},
  title = {Inscriptis - A Python-based HTML to text conversion library optimized for␣
↪knowledge extraction from the Web},
  journal = {Journal of Open Source Software}
}
```

11.1.10 Changelog

A full list of changes can be found in the release notes.

11.2 Testing, benchmarking and evaluation

11.2.1 Unit tests

In addition to the standard unit tests that are located in the project's *test* directory Inscriptis also contains test cases that solely focus on the html to text conversion and are located in the *tests/html* directory. These tests consist of two files:

1. *test-name.html* and
2. *test-name.txt*

The *.txt* file contains the reference text output for the given html file.

Since Inscripits 2.0 there may also be a third file named *test-name.json* in the *tests/html* directory which contains a JSON dictioanry with keys

1. *annotation-rules* containing the annotation rules for extracting metadata from the corresponding html file, and
2. *result* which stores the surface forms of the extracted metadata.

Example:

```
{"annotation_rules": {
    "h1": ["heading"],
    "b": ["emphasis"]
 },
 "result": [
        ["heading", "The first"],
        ["heading", "The second"],
        ["heading", "Subheading"]
 ]
}
```

11.2.2 Text conversion output comparison and benchmarking

The inscriptis project contains a benchmarking script that can compare different HTML to text conversion approaches. The script will run the different approaches on a list of URLs, *url_list.txt*, and save the text output into a time stamped folder in *benchmarking/benchmarking_results* for manual comparison. Additionally the processing speed of every approach per URL is measured and saved in a text file called *speed_comparisons.txt* in the respective time stamped folder.

To run the benchmarking script execute *run_benchmarking.py* from within the folder *benchmarking*. In *def pipeline()* set the which HTML -> Text algorithms to be executed by modifying:

```
run_lynx = True
run_justext = True
run_html2text = True
run_beautifulsoup = True
run_inscriptis = True
```

In *url_list.txt* the URLs to be parsed can be specified by adding them to the file, one per line with no additional formatting. URLs need to be complete (including http:// or https://) e.g.:

```
http://www.informationscience.ch
https://en.wikipedia.org/wiki/Information_science
...
```

11.3 Contributing to Inscriptis

First off, thank you for considering contributing to inscriptis. There are many ways how you can contribute to the project and these guidelines aim at supporting you in doing so.

1. *Reporting bugs and seeking support*
2. *Suggesting enhancements*
3. *Pull requests* (contributing code)
4. *Python style guide*

11.3.1 Reporting bugs and seeking support

Bugs and support requests are tracked as GitHub issues.

To create an effective and high quality ticket, please include the following information in your ticket:

1. **Use a clear and descriptive title** for the issue to identify the problem. This also helps other users to quickly locate bug reports that affect them.
2. **Describe the exact steps necessary for reproducing the problem** including at least information on
   - the affected URL
   - the command line parameters or function arguments you used
3. What would have been the **expected behavior**?
4. Describe the **observed behavior**.
5. Provide any additional information which might be helpful in reproducing and/or fixing this issue.

11.3.2 Suggesting enhancements

Enhancements are also tracked as GitHub issues and should contain the following information:

1. **A clear and descriptive title** helps other people to identify enhancements they like, so that they can also add their thoughts and suggestions.
2. **Provide a step-by-step description** of the suggested enhancement.
3. **Describe the current behavior** and **explain which behavior you expected to see instead** and why.

11.3.3 Pull requests

1. Ensure that your code complies with our *Python style guide*.
2. Write a unit test that covers your new code and put it into the `./tests` directory.
3. Execute `tox .` in the project's root directory to ensure that your code passes the static code analysis, coding style guidelines and security checks.
4. In addition, please document any new API functions in the Inscriptis documentation.

11.3.4 Python style guide

Inscriptis code should comply to

- the PEP8 Style Guide for Python Code, and
- to the Google Python Style Guide

Please also ensure that

1. functions are properly documented with docstrings that comply to the Google Python Style Guide, and
2. any new code is covered by unit tests.

11.4 Inscriptis module documentation

Parse HTML content and converts it into a text representation.

Inscriptis provides support for

- nested HTML tables
- basic Cascade Style Sheets
- annotations

The following example provides the text representation of https://www.fhgr.ch.

```
import urllib.request
from inscriptis import get_text

url = 'https://www.fhgr.ch'
html = urllib.request.urlopen(url).read().decode('utf-8')

text = get_text(html)
```

(continues on next page)

(continued from previous page)

```
print(text)
```

Use the method *get_annotated_text()* to obtain text and annotations. The method requires annotation rules as described in *annotations*.

```
import urllib.request
from inscriptis import get_annotated_text

url = "https://www.fhgr.ch"
html = urllib.request.urlopen(url).read().decode('utf-8')

# annotation rules specify the HTML elements and attributes to annotate.
rules = {'h1': ['heading'],
         'h2': ['heading'],
         '#class=FactBox': ['fact-box'],
         'i': ['emphasis']}

output = get_annotated_text(html, ParserConfig(annotation_rules=rules)
print("Text:", output['text'])
print("Annotations:", output['label'])
```

The method returns a dictionary with two keys:

1. *text* which contains the page's plain text and
2. ***label* with the annotations in JSONL format that is used by annotators**
   such as doccano.

Annotations in the *label* field are returned as a list of triples with
start index, *end index* and *label* as indicated below:

```
{"text": "Chur\n\nChur is the capital and largest town of the Swiss canton
          of the Grisons and lies in the Grisonian Rhine Valley.",
 "label": [[0, 4, "heading"], [6, 10, "emphasis"]]}
```

inscriptis.**get_annotated_text**(*html_content: str*, *config: ParserConfig = None*) → Dict[str, Any]

Return a dictionary of the extracted text and annotations.

Notes

- the text is stored under the key 'text'.
- annotations are provided under the key 'label' which contains a list of :class:`Annotation`s.

Examples

{"text": "EU rejects German call to boycott British lamb.", "
label": [[0, 2, "strong"], ...]}

{"text": "Peter Blackburn",
"label": [[0, 15, "heading"]]}

Returns
'text') and annotations (key: 'label')

Return type
A dictionary of text (key

inscriptis.**get_text**(*html_content: str*, *config: ParserConfig = None*) → str

Provide a text representation of the given HTML content.

Parameters

- **html_content** (*str*) – The HTML content to convert.
- **config** – An optional ParserConfig object.

Returns
The text representation of the HTML content.

11.4.1 Inscriptis model

Inscriptis HTML engine

The HTML Engine is responsible for converting HTML to text.

class inscriptis.html_engine.**Inscriptis**(*html_tree: HtmlElement*, *config: ParserConfig = None*)

Translate an lxml HTML tree to the corresponding text representation.

Parameters

- **html_tree** – the lxml HTML tree to convert.
- **config** – an optional ParserConfig configuration object.

Example:

```
from lxml.html import fromstring
from inscriptis.html_engine import Inscriptis

html_content = "<html><body><h1>Test</h1></body></html>"

# create an HTML tree from the HTML content.
html_tree = fromstring(html_content)
```

(continues on next page)

(continued from previous page)

```
# transform the HTML tree to text.
parser = Inscriptis(html_tree)
text = parser.get_text()
```

get_annotations() → List[*Annotation*]

Return the annotations extracted from the HTML page.

get_text() → str

Return the text extracted from the HTML page.

Inscriptis HTML properties

Provide properties used for rendering HTML pages.

Supported attributes::

1. *Display* properties.
2. *WhiteSpace* properties.
3. *HorizontalAlignment* properties.
4. *VerticalAlignment* properties.

class inscriptis.html_properties.**Display**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Specify whether content will be rendered as inline, block or none.

Note: A display attribute on none indicates, that the content should not be rendered at all.

class inscriptis.html_properties.**HorizontalAlignment**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Specify the content's horizontal alignment.

center = '^'

Center the block's content.

left = '<'

Left alignment of the block's content.

right = '>'

Right alignment of the block's content.

class inscriptis.html_properties.**VerticalAlignment**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Specify the content's vertical alignment.

bottom = 3

Align all content at the bottom.

middle = 2

Align all content in the middle.

top = 1

Align all content at the top.

class inscriptis.html_properties.**WhiteSpace**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Specify the HTML element's whitespace handling.

Inscriptis supports the following handling strategies outlined in the Cascading Style Sheets specification.

normal = 1

Collapse multiple whitespaces into a single one.

pre = 3

Preserve sequences of whitespaces.

Inscriptis CSS model

Implement basic CSS support for inscriptis.

- The `HtmlElement` class encapsulates all CSS properties of a single HTML element.
- *CssParse* parses CSS specifications and translates them into the corresponding HtmlElements used by Inscriptis for rendering HTML pages.

class inscriptis.model.css.**CssParse**

Parse CSS specifications and applies them to HtmlElements.

The attribute *display: none*, for instance, is translated to `HtmlElement.display=Display.none`.

static attr_display(*value: str*, *html_element: HtmlElement*)

Apply the given display value.

static attr_horizontal_align(*value: str*, *html_element: HtmlElement*)

Apply the provided horizontal alignment.

static attr_margin_after(*value: str*, *html_element: HtmlElement*)

Apply the provided bottom margin.

static attr_margin_before(*value: str*, *html_element: HtmlElement*)

Apply the given top margin.

static attr_margin_bottom(*value: str*, *html_element: HtmlElement*)

Apply the provided bottom margin.

static attr_margin_top(*value: str*, *html_element: HtmlElement*)

Apply the given top margin.

static attr_padding_left(*value: str*, *html_element: HtmlElement*)

Apply the given left padding_inline.

static attr_padding_start(*value: str*, *html_element: HtmlElement*)

Apply the given left padding_inline.

static attr_style(*style_attribute: str*, *html_element: HtmlElement*)

Apply the provided style attributes to the given HtmlElement.

Parameters

- **style_attribute** – The attribute value of the given style sheet. Example: display: none

- **html_element** – The HtmlElement to which the given style is applied.

static attr_vertical_align(*value: str*, *html_element: HtmlElement*)

Apply the given vertical alignment.

static attr_white_space(*value: str*, *html_element: HtmlElement*)

Apply the given white-space value.

Inscriptis canvas model

Classes used for rendering (parts) of the canvas.

Every parsed `HtmlElement` writes its textual content to the canvas which is managed by the following three classes:

- *Canvas* provides the drawing board on which the HTML page is serialized and annotations are recorded.
- *Block* contains the current line to which text is written.
- *Prefix* handles indentation and bullets that prefix a line.

class inscriptis.model.canvas.**Canvas**

The text Canvas on which Inscriptis writes the HTML page.

margin

the current margin to the previous block (this is required to ensure that the *margin_after* and *margin_before* constraints of HTML block elements are met).

current_block

A *Block* which merges the input text into a block (i.e., line).

blocks

a list of strings containing the completed blocks (i.e., text lines). Each block spawns at least one line.

annotations

the list of recorded *Annotation*s.

_open_annotations

a map of open tags that contain annotations.

close_block(*tag: HtmlElement*) → None

Close the given HtmlElement by writing its bottom margin.

Parameters
tag – the HTML Block element to close

close_tag(*tag: HtmlElement*) → None

Register that the given tag tag is closed.

Parameters
tag – the tag to close.

flush_inline() → bool

Attempt to flush the content in self.current_block into a new block.

Notes

- If self.current_block does not contain any content (or only whitespaces) no changes are made.
- Otherwise the content of current_block is added to blocks and a new current_block is initialized.

Returns
True if the attempt was successful, False otherwise.

get_text() → str
Provide a text representation of the Canvas.

property left_margin: int
Return the length of the current line's left margin.

open_block(*tag: HtmlElement*) → None
Open an HTML block element.

open_tag(*tag: HtmlElement*) → None
Register that a tag is opened.

Parameters
tag – the tag to open.

write(*tag: HtmlElement*, *text: str*, *whitespace:* WhiteSpace = *None*) → None
Write the given text to the current block.

write_unconsumed_bullet() → None
Write unconsumed bullets to the blocks list.

Representation of a text block within the HTML canvas.

class inscriptis.model.canvas.block.**Block**(*idx: int*, *prefix:* Prefix)
The current block of text.

A block usually refers to one line of output text.

Note: If pre-formatted content is merged with a block, it may also contain multiple lines.

Parameters

- **idx** – the current block's start index.
- **prefix** – prefix used within the current block.

merge(*text: str*, *whitespace:* WhiteSpace) → None
Merge the given text with the current block.

Parameters

- **text** – the text to merge.
- **whitespace** – whitespace handling.

merge_normal_text(*text: str*) → None
Merge the given text with the current block.

Parameters
text – the text to merge

Note:

If the previous text ended with a whitespace and text starts with one, both
will automatically collapse into a single whitespace.

merge_pre_text(*text: str*) → None

Merge the given pre-formatted text with the current block.

Parameters
text – the text to merge

new_block() → *Block*

Return a new Block based on the current one.

Manage the horizontal prefix (left-indentation, bullets) of canvas lines.

class inscriptis.model.canvas.prefix.**Prefix**

Class Prefix manages paddings and bullets that prefix an HTML block.

current_padding

the number of characters used for the current left-indentation.

paddings

the list of paddings for the current and all previous tags.

bullets

the list of bullets in the current and all previous tags.

consumed

whether the current bullet has already been consumed.

property first: str

Return the prefix used at the beginning of a tag.

Note::
A new block needs to be prefixed by the current padding and bullet. Once this has happened (i.e., *consumed* is set to *True*) no further prefixes should be used for a line.

pop_next_bullet() → str

Pop the next bullet to use, if any bullet is available.

register_prefix(*padding_inline: int*, *bullet: str*) → None

Register the given prefix.

Parameters

- **padding_inline** – the number of characters used for padding_inline
- **bullet** – an optional bullet.

remove_last_prefix() → None

Remove the last prefix from the list.

property rest: str

Return the prefix used for new lines within a block.

This prefix is used for pre-text that contains newlines. The lines need to be prefixed with the right padding to preserver the indentation.

property unconsumed_bullet: str

Yield any yet unconsumed bullet.

Note::
This function yields the previous element's bullets, if they have not been consumed yet.

Inscriptis table model

Classes used for representing Tables, TableRows and TableCells.

class inscriptis.model.table.**Table**(*left_margin_len: int*, *cell_separator: str*)

An HTML table.

rows

the table's rows.

left_margin_len

length of the left margin before the table.

cell_separator

string used for separating cells from each other.

add_cell(*table_cell:* TableCell)

Add a new *TableCell* to the table's last row.

Note: If no row exists yet, a new row is created.

---

add_row()

Add an empty *TableRow* to the table.

get_annotations(*idx: int*, *left_margin_len: int*) → List[*Annotation*]

Return all annotations in the given table.

Parameters

- **idx** – the table's start index.
- **left_margin_len** – len of the left margin (required for adapting the position of annotations).

Returns
A list of all *Annotation*s present in the table.

get_text() → str

Return and render the text of the given table.

class inscriptis.model.table.**TableCell**(*align:* HorizontalAlignment, *valign:* VerticalAlignment)

A table cell.

line_width

the original line widths per line (required to adjust annotations after a reformatting)

vertical_padding

vertical padding that has been introduced due to vertical formatting rules.

get_annotations(*idx: int*, *row_width: int*) → List[*Annotation*]

Return a list of all annotations within the TableCell.

Returns
A list of annotations that have been adjusted to the cell's position.

property height: int

Compute the table cell's height.

Returns
The cell's current height.

normalize_blocks() → int

Split multi-line blocks into multiple one-line blocks.

Returns
The height of the normalized cell.

property width: int

Compute the table cell's width.

Returns
The cell's current width.

class inscriptis.model.table.**TableRow**(*cell_separator: str*)

A single row within a table.

columns

the table row's columns.

cell_separator

string used for separating columns from each other.

get_text() → str

Return a text representation of the TableRow.

property width: int

Compute and return the width of the current row.

11.4.2 Inscriptis annotations

The model used for saving annotations.

class inscriptis.annotation.**Annotation**(*start: int*, *end: int*, *metadata: str*)

An Inscriptis annotation which provides metadata on the extracted text.

The *start* and *end* indices indicate the span of the text to which the metadata refers, and the attribute *metadata* contains the tuple of tags describing this span.

Example:

```
Annotation(0, 10, ('heading', ))
```

The annotation above indicates that the text span between the 1st (index 0) and 11th (index 10) character of the extracted text contains a *heading*.

end: int

the annotation's end index within the text output.

metadata: str

the tag to be attached to the annotation.

start: int

the annotation's start index within the text output.

inscriptis.annotation.**horizontal_shift**(*annotations: List[*Annotation*]*, *content_width: int*, *line_width: int*, *align:* HorizontalAlignment, *shift: int = 0*) → List[*Annotation*]

Shift annotations based on the given line's formatting.

Adjusts the start and end indices of annotations based on the line's formatting and width.

Parameters

- **annotations** – a list of Annotations.
- **content_width** – the width of the actual content
- **line_width** – the width of the line in which the content is placed.
- **align** – the horizontal alignment (left, right, center) to assume for the adjustment
- **shift** – an optional additional shift

Returns

A list of *Annotation*s with the adjusted start and end positions.

Annotation processors

*AnnotationProcessor*s transform annotations to an output format.

All AnnotationProcessor's implement the *AnnotationProcessor* interface by overwrite the class's `AnnotationProcessor.__call__()` method.

Note:

1. The AnnotationExtractor class must be put into a package with the extractor's name (e.g., `inscriptis.annotation.output.*package*`) and be named `*PackageExtractor*` (see the examples below).
2. The overwritten `__call__()` method may either extend the original dictionary which contains the extracted text and annotations (e.g., `SurfaceExtractor`) or may replace it with an custom output (e.g., `HtmlExtractor` and `XmlExtractor`.

Currently, Inscriptis supports the following built-in AnnotationProcessors:

1. `HtmlExtractor` provides an annotated HTML output format.
2. `XmlExtractor` yields an output which marks annotations with XML tags.
3. `SurfaceExtractor` adds the key *surface* to the result dictionary which contains the surface forms of the extracted annotations.

class inscriptis.annotation.output.**AnnotationProcessor**

An AnnotationProcessor is called for formatting annotations.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W