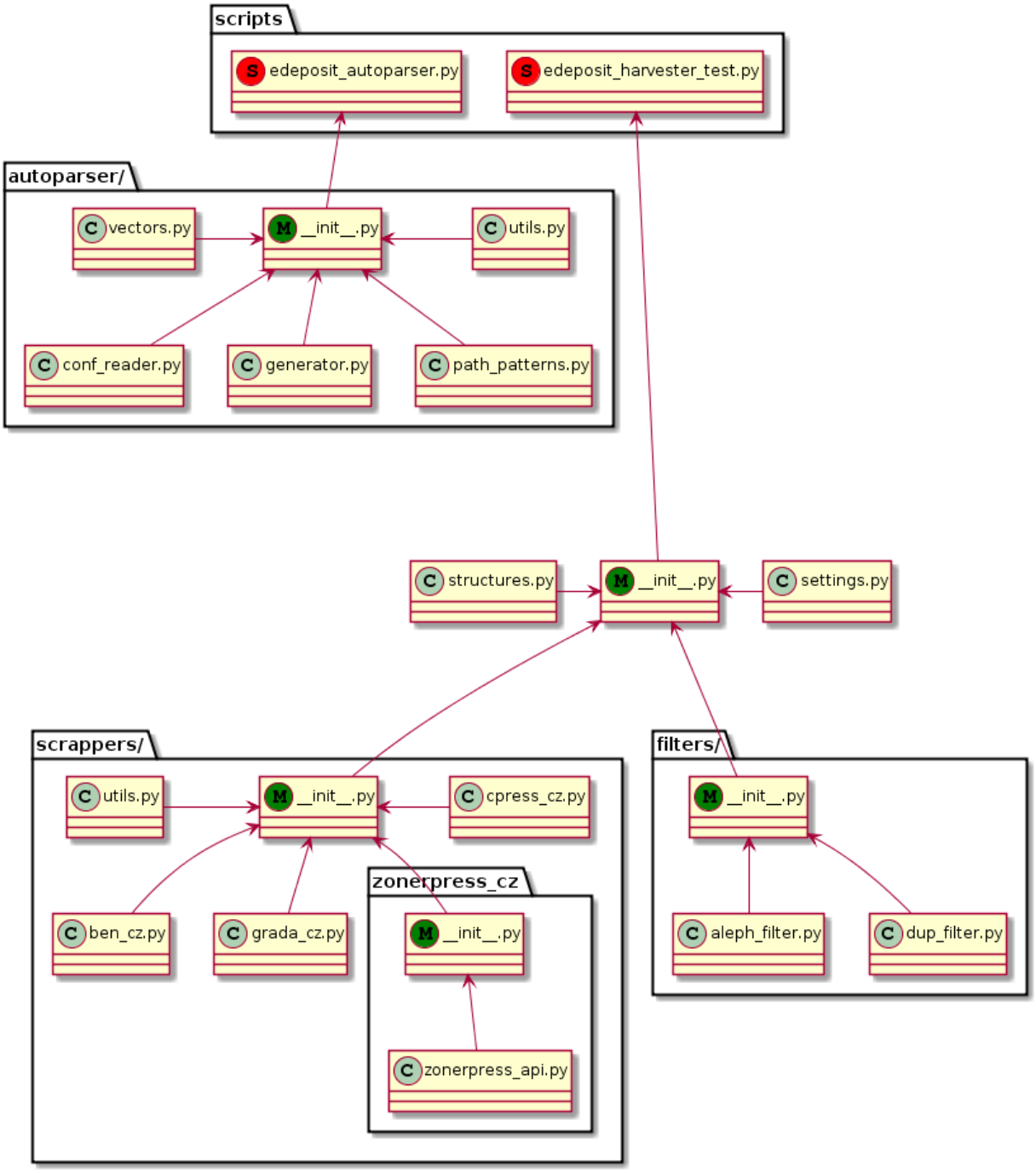
edeposit.amqp.harvester

Release 1.0.1

September 10, 2015

Contents

This module is used to collect public metadata about new books published by selected czech publishers.



CHAPTER 1

User guide / Uživatelská příručka

1.1 edeposit.amqp.harvester

Tento modul obsahuje funkce pro stahování metadat ze stránek několika vybraných vydavatelů. Momentálně jsou k dispozici programové komponenty pro webové prezentace nakladatelství Ben, Grada, CPress a ZonerPress.

1.1.1 Instalace modulu

Modul je možné nainstalovat na prakticky každý linuxový systém pomocí programu pip, který je součástí standardní distribuce *pythonu*:

```
sudo pip install edeposit.amqp.harvester
```

1.1.2 Použití modulu

Podobně jako ostatní prvky projektu Edeposit je i tento modul součástí asynchronního distribuovaného systému, jehož jednotlivé komponenty spolu komunikují přes AMQP protokol. O to se stará modul edeposit.amqp.

edeposit.amqp.harvester poskytuje pouze rozhraní umožňující *sklízení* metadat, nikoliv script, které získané informace předává dál. Ten je možné najít v modulu edeposit.amqp, kde se nachází pod názvem edeposit_amqp_harvester.py.

Spuštěním tohoto scriptu dochází k "sklizení" dat ze všech podporovaných komponent a jejich odeslání na AMQP fronty tak, jak je to definováno v souboru `settingy.py` modulu *edeposit.amqp*. Data jsou odesílána ve formátu struktury `Publications`, která ve svém těle nese pole struktur `Publication` se sklizenými metadaty.

Filtrace dat

Modul umožňuje a v základu používá filtraci již zpracovaných záznamů. V tomto režimu jsou všechny stažené výsledky porovnávány vůči lokální databázi (viz soubor definovaný v `harvester.settings.DUP_FILTER_FILE`) a odesílány jsou pouze ty, které ještě nebyly zpracovány.

Toto chování je možné změnit nastavením konfigurační proměnné `harvester.settings.USE_DUP_FILTER` na hodnotu `False`.

Dostupný je také filtr, který výsledky porovnává vůči Alephu a propouští pouze ty záznamy, které zatím Aleph neobsahuje.

Tento filtr je v základě vypnut aby se předešlo zbytečné zátěži Alephu. Zapnout toto chování je možné nastavením konfigurační proměnné `harvester.settings.USE_ALEPH_FILTER` na hodnotu `True`.

1.1.3 Testovací script

Pro potřeby uživatelského testování byl v modulu *edeposit.amqp.harvester* vytvořen testovací script, který "sklidí" všechna data a zobrazí je na standardní výstup.

Script je možné najít ve složce `bin/` pod názvem `edeposit_harvester_test.py`.

Zde je ukázka nápovědy:

```
$ ./edeposit_harvester_test.py -h
usage: edeposit_harvester_test.py [-h] [-u] [-r]

This script is used to read data from edeposit.amqp.harvester and print it to
stdout.

optional arguments:
  -h, --help      show this help message and exit
  -u, --unittest  Perform unittest.
  -r, --harvest   Harvest all data and send them to harvester queue.
```

Jak je vidět z nápovědy, script přijímá dva parametry `--unittest` pro spuštění testu jednotlivých komponent pro sklízení dat a `--harvest`, jenž stáhne všechna dostupná data a vypíše je na standardní výstup.

Výsledek spuštění s parametrem `--harvest` je možné najít například zde:

- `_static/out.txt`

Stejná data jsou normálně odeslána přes AMQP.

1.1.4 Testování modulu

Všechny komponenty, které má smysl automaticky testovat jsou testovány scriptem `run_tests.sh`, který se nachází v kořenovém adresáři projektu.

Tento script je postavený nad programem py.test, jenž je možné nainstalovat příkazem:

```
sudo pip install pytest
```

Zde je ukázka běhu všech 116 testů:

```
$ ./run_tests.sh -u
============================= test session starts ==============================
platform linux2 -- Python 2.7.5 -- py-1.4.20 -- pytest-2.5.2
collected 116 items

src/edeposit/amqp/harvester/tests/unittests/test_aleph_filter_unit.py ..
src/edeposit/amqp/harvester/tests/unittests/test_autoparser.py .........
src/edeposit/amqp/harvester/tests/unittests/test_dup_filter.py ...
src/edeposit/amqp/harvester/tests/unittests/test_settings.py .
src/edeposit/amqp/harvester/tests/unittests/test_structures.py ....
src/edeposit/amqp/harvester/tests/unittests/autoparser/test_auto_utils.py ....
src/edeposit/amqp/harvester/tests/unittests/autoparser/test_conf_reader.py ...
src/edeposit/amqp/harvester/tests/unittests/autoparser/test_path_patterns.py ............
src/edeposit/amqp/harvester/tests/unittests/autoparser/test_vectors.py ...
src/edeposit/amqp/harvester/tests/unittests/scrappers/test_ben_cz.py ..............................
src/edeposit/amqp/harvester/tests/unittests/scrappers/test_cpress_cz.py ..................
src/edeposit/amqp/harvester/tests/unittests/scrappers/test_grada_cz.py ..........
src/edeposit/amqp/harvester/tests/unittests/scrappers/test_utils.py ........

========================== 116 passed in 2.65 seconds ==========================
```

CHAPTER 2

Scripts

2.1 edeposit_autoparser.py

This script is used to ease creation of new parsers.

2.1.1 Configuration file

The script expects configuration file with patterns, specified as `-c` parameter. Pattern files uses YAML as serialization format.

Inside the pattern file should be multiple pattern definitions. Here is example of the test pattern file:

```
html: simple_xml.xml
first:
    data: i wan't this
    required: true
    notfoundmsg: Can't find variable '$name'.
second:
    data: and this
---
html: simple_xml2.xml
first:
    data: something wanted
    required: true
    notfoundmsg: Can't find variable '$name'.
second:
    data: another wanted thing
```

As you can see, this file contains two examples divided by `---`. Each section, of file have to contain `html` key pointing to either file or URL resource.

After the `html` key, there may be unlimited number of *variables*. Each *variable* have to contain `data` key, which defines the match, which will be parsed from the file `html` key is pointing to.

Optionally, you can also specify `required` and `notfoundmsg`. If the variable is required, it means that if generated parser will found data without this variable, UserWarning exception is raised and `notfoundmsg` is used as message. As you can see in example, you can use `$name` as variable which holds variable name (*first* for example).

There is also special keyword `tagname`, which can be used to further specify correct element in case, that there is more than one element matching.

2.1.2 How it works

Autoparser first reads all examples and locates elements, which content matching pattern defined in `data` key. Spaces at the beginning and end of the pattern and element's content are ignored.

When the autoparser collects all matching elements, it generates *DOM* paths to each element.

After that, elimination process begins. In this step, autoparser throws away all paths, that doesn't work for all corresponding variables in all examples.

When this is done, paths with best priority are selected and `generate_parsers()` is called.

Result from this call is string printed to the output. This string contains all necessary parsers for each variable and also unittest.

You can then build the parser you need much more easilly, because now you have working *pickers* from *DOM* and all you need to do is to clean the data.

Live example:

```
$ ./edeposit_autoparser.py -c autoparser/autoparser_data/example_data.yaml
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#
# Interpreter version: python 2.7
#
# HTML parser generated by Autoparser
# (https://github.com/edeposit/edeposit.amqp.harvester)
#
import os
import os.path

import httpkie
import dhtmlparser


# Utilities
def _get_source(link):
    """
    Return source of the `link` whether it is filename or url.

    Args:
        link (str): Filename or URL.

    Returns:
        str: Content.

    Raises:
        UserWarning: When the `link` couldn't be resolved.
    """
    if link.startswith("http://") or link.startswith("https://"):
        down = httpkie.Downloader()
        return down.download(link)

    if os.path.exists(link):
        with open(link) as f:
            return f.read()

    raise UserWarning("html: '%s' is neither URL or data!" % link)
```

```
def _get_encoding(dom, default="utf-8"):
    """
    Try to look for meta tag in given `dom`.

    Args:
        dom (obj): pyDHTMLParser dom of HTML elements.
        default (default "utr-8"): What to use if encoding is not found in
                                   `dom`.

    Returns:
        str/default: Given encoding or `default` parameter if not found.
    """
    encoding = dom.find("meta", {"http-equiv": "Content-Type"})

    if not encoding:
        return default

    encoding = encoding[0].params.get("content", None)

    if not encoding:
        return default

    return encoding.lower().split("=")[-1]


def handle_encodnig(html):
    """
    Look for encoding in given `html`. Try to convert `html` to utf-8.

    Args:
        html (str): HTML code as string.

    Returns:
        str: HTML code encoded in UTF.
    """
    encoding = _get_encoding(
        dhtmlparser.parseString(
            html.split("</head>")[0]
        )
    )

    if encoding == "utf-8":
        return html

    return html.decode(encoding).encode("utf-8")


def is_equal_tag(element, tag_name, params, content):
    """
    Check is `element` object match rest of the parameters.

    All checks are performed only if proper attribute is set in the HTMLElement.

    Args:
        element (obj): HTMLElement instance.
        tag_name (str): Tag name.
        params (dict): Parameters of the tag.
        content (str): Content of the tag.
```

```
    Returns:
        bool: True if everyhing matchs, False otherwise.
    """
    if tag_name and tag_name != element.getTagName():
        return False

    if params and not element.containsParamSubset(params):
        return False

    if content is not None and content.strip() != element.getContent().strip():
        return False

    return True


def has_neigh(tag_name, params=None, content=None, left=True):
    """
    This function generates functions, which matches all tags with neighbours
    defined by parameters.

    Args:
        tag_name (str): Tag has to have neighbour with this tagname.
        params (dict): Tag has to have neighbour with this parameters.
        params (str): Tag has to have neighbour with this content.
        left (bool, default True): Tag has to have neigbour on the left, or
                                   right (set to ``False``).

    Returns:
        bool: True for every matching tag.

    Note:
        This function can be used as parameter for ``.find()`` method in
        HTMLElement.
    """
    def has_neigh_closure(element):
        if not element.parent \
           or not (element.isTag() and not element.isEndTag()):
            return False

        # filter only visible tags/neighbours
        childs = element.parent.childs
        childs = filter(
            lambda x: (x.isTag() and not x.isEndTag()) \
                      or x.getContent().strip() or x is element,
            childs
        )
        if len(childs) <= 1:
            return False

        ioe = childs.index(element)
        if left and ioe > 0:
            return is_equal_tag(childs[ioe - 1], tag_name, params, content)

        if not left and ioe + 1 < len(childs):
            return is_equal_tag(childs[ioe + 1], tag_name, params, content)

        return False
```

```
    return has_neigh_closure


# Generated parsers
def get_second(dom):
    el = dom.find(
        'container',
        {'id': 'mycontent'},
        fn=has_neigh(None, None, 'something something', left=False)
    )

    # pick element from list
    el = el[0] if el else None

    return el


def get_first(dom):
    el = dom.wfind('root').childs

    if not el:
        raise UserWarning(
            "Can't find variable 'first'.\n" +
            'Tag name: root\n' +
            'El:' + str(el) + '\n' +
            'Dom:' + str(dom)
        )

    el = el[-1]

    el = el.wfind('xax').childs

    if not el:
        raise UserWarning(
            "Can't find variable 'first'.\n" +
            'Tag name: xax\n' +
            'El:' + str(el) + '\n' +
            'Dom:' + str(dom)
        )

    el = el[-1]

    el = el.wfind('container').childs

    if not el:
        raise UserWarning(
            "Can't find variable 'first'.\n" +
            'Tag name: container\n' +
            'El:' + str(el) + '\n' +
            'Dom:' + str(dom)
        )

    el = el[-1]

    return el


# Unittest
```

```
def test_parsers():
    # Test parsers against autoparser/autoparser_data/simple_xml.xml
    html = handle_encodnig(
        _get_source('autoparser/autoparser_data/simple_xml.xml')
    )
    dom = dhtmlparser.parseString(html)
    dhtmlparser.makeDoubleLinked(dom)

    second = get_second(dom)
    assert second.getContent().strip() == 'and this'

    first = get_first(dom)
    assert first.getContent().strip() == "i wan't this"

    # Test parsers against autoparser/autoparser_data/simple_xml2.xml
    html = handle_encodnig(
        _get_source('autoparser/autoparser_data/simple_xml2.xml')
    )
    dom = dhtmlparser.parseString(html)
    dhtmlparser.makeDoubleLinked(dom)

    second = get_second(dom)
    assert second.getContent().strip() == 'another wanted thing'

    first = get_first(dom)
    assert first.getContent().strip() == 'something wanted'


# Run tests of the parser
if __name__ == '__main__':
    test_parsers()
```

2.1.3 API

harvester.edeposit_autoparser.**_create_dom**(*data*)

Creates doublelinked DOM from *data*.

Parameters **data** (*str/HTMLElement*) – Either string or HTML element.

Returns HTMLElement containing double linked DOM.

Return type obj

harvester.edeposit_autoparser.**_locate_element**(*dom*, *el_content*, *transformer=None*)

Find element containing *el_content* in *dom*. Use *transformer* function to content of all elements in *dom* in order to correctly transforming them to match them with *el_content*.

Parameters

- **dom** (*obj*) – HTMLElement tree.
- **el_content** (*str*) – Content of element will be picked from *dom*.
- **transformer** (*fn, default None*) – Transforming function.

Note: *transformer* parameter can be for example simple lambda:

```
lambda x: x.strip()
```

Returns Matching HTMLElements.

Return type list

harvester.edeposit_autoparser.**_match_elements**(*dom*, *matches*)
Find location of elements matching patterns specified in *matches*.

Parameters

- **dom** (*obj*) – HTMLElement DOM tree.
- **matches** (*dict*) – Structure: `{"var": {"data": "match", ..}, ..}`.

Returns Structure: `{"var": {"data": HTMLElement_obj, ..}, ..}`

Return type dict

harvester.edeposit_autoparser.**_collect_paths**(*element*)
Collect all possible path which leads to *element*.

Function returns standard path from root element to this, reverse path, which uses negative indexes for path, also some pattern matches, like "this is element, which has neighbour with id 7" and so on.

Parameters **element** (*obj*) – HTMLElement instance.

Returns List of `PathCall` and `Chained` objects.

Return type list

harvester.edeposit_autoparser.**_is_working_path**(*dom*, *path*, *element*)
Check whether the path is working or not.

Aply proper search function interpreting *path* to *dom* and check, if returned object is *element*. If so, return `True`, otherwise `False`.

Parameters

- **dom** (*obj*) – HTMLElement DOM.
- **path** (*obj*) – `PathCall` Instance containing informations about path and which function it require to obtain element the path is pointing to.
- **element** (*obj*) – HTMLElement instance used to decide whether *path* points to correct *element* or not.

Returns True if *path* correctly points to proper *element*.

Return type bool

harvester.edeposit_autoparser.**select_best_paths**(*examples*)
Process *examples*, select only paths that works for every example. Select best paths with highest priority.

Parameters **examples** (*dict*) – Output from `read_config()`.

Returns List of `PathCall` and `Chained` objects.

Return type list

2.2 edeposit_harvester_test.py

Test script used to show output of all downloaded data.

Help:

```
$ ./edeposit_harvester_test.py -h
usage: edeposit_harvester_test.py [-h] [-u] [-r] [-d]

This script is used to read data from edeposit.amqp.harvester and print it to
stdout.

optional arguments:
  -h, --help        show this help message and exit
  -u, --unittest    Perform unittest.
  -r, --harvest     Harvest all data and send them to harvester queue.
  -d, --dup-filter  Filter duplicate results. Default False.
```

2.2.1 API

edeposit_harvester_test.**print_messages**(*pubs*)

Print all publications from *pubs*.

CHAPTER 3

API

Whole module is divided into following parts:

3.1 Scrappers

`Scrappers` are used to download metadata from publisher's webpages.

3.1.1 ben.cz scrapper

This module is used to download last 100 books published by *ben.cz*.

`harvester.scrappers.ben_cz.`**`URL = 'http://shop.ben.cz/Produkty.aspx?lang=cz&nak=BEN+-+technick%u00e1+literat`**
Base url of the eshop.

`harvester.scrappers.ben_cz.`**`_get_last_td`**(*el*)
Return last <td> found in *el* DOM.

Parameters **el** (*obj*) – `dhtmlparser.HTMLElement` instance.

Returns HTMLElement instance if found, or None if there are no <td> tags.

Return type obj

`harvester.scrappers.ben_cz.`**`_get_td_or_none`**(*details*, *ID*)
Get <tr> tag with given *ID* and return content of the last <td> tag from <tr> root.

Parameters

- **details** (*obj*) – `dhtmlparser.HTMLElement` instance.
- **ID** (*str*) – id property of the <tr> tag.

Returns Content of the last <td> as strign.

Return type str

`harvester.scrappers.ben_cz.`**`_parse_title`**(*dom*, *details*)
Parse title/name of the book.

Parameters

- **dom** (*obj*) – HTMLElement containing whole HTML page.
- **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns Book's title.

Return type str

Raises `AssertionError` – If title not found.

`harvester.scrappers.ben_cz._parse_authors`(*details*)
Parse authors of the book.

Parameters **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns List of `structures.Author` objects. Blank if no author found.

Return type list

`harvester.scrappers.ben_cz._parse_publisher`(*details*)
Parse publisher of the book.

Parameters **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns Publisher's name as string or None if not found.

Return type str/None

`harvester.scrappers.ben_cz._parse_price`(*details*)
Parse price of the book.

Parameters **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns Price as string with currency or None if not found.

Return type str/None

`harvester.scrappers.ben_cz._parse_pages_binding`(*details*)
Parse number of pages and binding of the book.

Parameters **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns Tuple with two string or two None.

Return type (pages, binding)

`harvester.scrappers.ben_cz._parse_ISBN_EAN`(*details*)
Parse ISBN and EAN.

Parameters **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns Tuple with two string or two None.

Return type (ISBN, EAN)

`harvester.scrappers.ben_cz._parse_edition`(*details*)
Parse edition (vydání) of the book.

Parameters **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns Edition as string with currency or None if not found.

Return type str/None

`harvester.scrappers.ben_cz._parse_description`(*details*)
Parse description of the book.

Parameters **details** (*obj*) – HTMLElement containing slice of the page with details.

Returns Details as string with currency or None if not found.

Return type str/None

`harvester.scrappers.ben_cz._process_book`(*book_url*)
Parse available informations about book from the book details page.

Parameters **book_url** (*str*) – Absolute URL of the book.

Returns `structures.Publication` instance with book details.

Return type obj

`harvester.scrappers.ben_cz.`**`get_publications`**`()`
Get list of publication offered by ben.cz.

Returns List of `structures.Publication` objects.

Return type list

`harvester.scrappers.ben_cz.`**`self_test`**`()`
Perform basic selftest.

Returns When everything is ok.

Return type True

Raises `AssertionError` – When there is some problem.

3.1.2 cpress.cz scrapper

This module is used to download metadata informations from *cpress.cz*.

`harvester.scrappers.cpress_cz.`**`_parse_alt_title`**`(`*html_chunk*`)`
Parse title from alternative location if not found where it should be.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Book's title.

Return type str

`harvester.scrappers.cpress_cz.`**`_parse_alt_url`**`(`*html_chunk*`)`
Parse URL from alternative location if not found where it should be.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Book's URL.

Return type str

`harvester.scrappers.cpress_cz.`**`_parse_title_url`**`(`*html_chunk*`)`
Parse title/name of the book and URL of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns (title, url), both as strings.

Return type tuple

`harvester.scrappers.cpress_cz.`**`_parse_authors`**`(`*html_chunk*`)`
Parse authors of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns List of `structures.Author` objects. Blank if no author found.

Return type list

`harvester.scrappers.cpress_cz.`**`_parse_price`**`(`*html_chunk*`)`
Parse price of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Price as string with currency or None if not found.

Return type str/None

harvester.scrappers.cpress_cz.**_parse_from_table**(*html_chunk*, *what*)
Go thru table data in *html_chunk* and try to locate content of the neighbor cell of the cell containing *what*.

Returns Table data or None.

Return type str

harvester.scrappers.cpress_cz.**_parse_ean**(*html_chunk*)
Parse EAN.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns EAN as string or None if not found.

Return type str/None

harvester.scrappers.cpress_cz.**_parse_date**(*html_chunk*)
Parse date.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns date as string or None if not found.

Return type str/None

harvester.scrappers.cpress_cz.**_parse_format**(*html_chunk*)
Parse format.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Format as string or None if not found.

Return type str/None

harvester.scrappers.cpress_cz.**_parse_description**(*html_chunk*)
Parse description of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Description as string or None if not found.

Return type str/None

harvester.scrappers.cpress_cz.**_process_book**(*html_chunk*)
Parse available informations about book from the book details page.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns `structures.Publication` instance with book details.

Return type obj

harvester.scrappers.cpress_cz.**get_publications**()
Get list of publication offered by cpress.cz.

Returns List of `Publication` objects.

Return type list

harvester.scrappers.cpress_cz.**self_test**()
Perform basic selftest.

Returns When everything is ok.

Return type True

Raises `AssertionError` – When there is some problem.

3.1.3 grada.cz scrapper

This module is used to download metadata from *grada.cz*.

`harvester.scrappers.grada_cz.`**`_parse_alt_title`**(*html_chunk*)
Parse title from alternative location if not found where it should be.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Book's title.

Return type str

`harvester.scrappers.grada_cz.`**`_parse_title_url`**(*html_chunk*)
Parse title/name of the book and URL of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns (title, url), both as strings.

Return type tuple

`harvester.scrappers.grada_cz.`**`_parse_subtitle`**(*html_chunk*)
Parse subtitle of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Subtitle or None if subtitle wasn't found.

Return type str/None

`harvester.scrappers.grada_cz.`**`_parse_authors`**(*html_chunk*)
Parse authors of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns List of `structures.Author` objects. Blank if no author found.

Return type list

`harvester.scrappers.grada_cz.`**`_parse_description`**(*html_chunk*)
Parse description of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Details as string with currency or None if not found.

Return type str/None

`harvester.scrappers.grada_cz.`**`_parse_format_pages_isbn`**(*html_chunk*)
Parse format, number of pages and ISBN.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns (format, pages, isbn), all as string.

Return type tuple

`harvester.scrappers.grada_cz.`**`_parse_price`**(*html_chunk*)
Parse price of the book.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns Price as string with currency or None if not found.

Return type str/None

harvester.scrappers.grada_cz.**_process_book**(*html_chunk*)
Parse available informations about book from the book details page.

Parameters **html_chunk** (*obj*) – HTMLElement containing slice of the page with details.

Returns `structures.Publication` instance with book details.

Return type obj

harvester.scrappers.grada_cz.**get_publications**()
Get list of publication offered by grada.cz.

Returns List of `Publication` objects.

Return type list

harvester.scrappers.grada_cz.**self_test**()
Perform basic selftest.

Returns When everything is ok.

Return type True

Raises `AssertionError` – When there is some problem.

3.1.4 zonerpress_cz scrapper

Module for parsing informations from *zonerpress.cz*.

harvester.scrappers.zonerpress_cz.**_get_max_page**(*dom*)
Try to guess how much pages are in book listing.

Parameters **dom** (*obj*) – HTMLElement container of the page with book list.

Returns Number of pages for given category.

Return type int

harvester.scrappers.zonerpress_cz.**_parse_book_links**(*dom*)
Parse links to the details about publications from page with book list.

Parameters **dom** (*obj*) – HTMLElement container of the page with book list.

Returns List of strings / absolute links to book details.

Return type list

harvester.scrappers.zonerpress_cz.**get_book_links**(*links*)
Go thru *links* to categories and return list to all publications in all given categories.

Parameters **links** (*list*) – List of strings (absolute links to categories).

Returns List of strings / absolute links to book details.

Return type list

harvester.scrappers.zonerpress_cz.**_strip_content**(*el*)
Call `.getContent()` method of the *el* and strip whitespaces. Return `None` if content is `-`.

Parameters **el** (*obj*) – HTMLElement instance.

Returns Clean string.

Return type str/None

harvester.scrappers.zonerpress_cz.**_parse_authors**(*authors*)
Parse informations about authors of the book.

Parameters **dom** (*obj*) – HTMLElement containing slice of the page with details.

Returns List of `Author` objects. Blank if no author found.

Return type list

harvester.scrappers.zonerpress_cz.**_process_book**(*link*)
Download and parse available informations about book from the publishers webpages.

Parameters **link** (*str*) – URL of the book at the publishers webpages.

Returns `Publication` instance with book details.

Return type obj

harvester.scrappers.zonerpress_cz.**get_publications**()
Get list of publication offered by ben.cz.

Returns List of `structures.Publication` objects.

Return type list

harvester.scrappers.zonerpress_cz.**self_test**()
Perform basic selftest.

Returns When everything is ok.

Return type True

Raises `AssertionError` – When there is some problem.

3.1.5 utils submodule

This module contains number of functions, which are used in the rest of the scrappers submodule.

harvester.scrappers.utils.**_get_encoding**(*dom*, *default='utf-8'*)
Try to look for meta tag in given *dom*.

Parameters

- **dom** (*obj*) – pyDHTMLParser dom of HTML elements.
- **default** (*default "utr-8"*) – What to use if encoding is not found in *dom*.

Returns Given encoding or *default* parameter if not found.

Return type str/default

harvester.scrappers.utils.**handle_encodnig**(*html*)
Look for encoding in given *html*. Try to convert *html* to utf-8.

Parameters **html** (*str*) – HTML code as string.

Returns HTML code encoded in UTF.

Return type str

harvester.scrappers.utils.**get_first_content**(*el_list*, *alt=None*, *strip=True*)
Return content of the first element in *el_list* or *alt*. Also return *alt* if the content string of first element is blank.

Parameters

- **el_list** (*list*) – List of HTMLElement objects.

- **alt** (*default None*) – Value returner when list or content is blank.
- **strip** (*bool, default True*) – Call .strip() to content.

Returns String representation of the content of the first element or *alt* if not found.

Return type str or alt

harvester.scrappers.utils.**is_absolute_url**(*url*, *protocol='http'*)
Test whether *url* is absolute url (`http://domain.tld/something`) or relative (`../something`).

Parameters

- **url** (*str*) – Tested string.
- **protocol** (*str, default "http"*) – Protocol which will be seek at the beginning of the *url*.

Returns True if url is absolute, False if not.

Return type bool

harvester.scrappers.utils.**normalize_url**(*base_url*, *rel_url*)
Normalize the *url* - from relative, create absolute URL.

Parameters

- **base_url** (*str*) – Domain with `protocol://` string
- **rel_url** (*str*) – Relative or absolute url.

Returns Normalized URL or None if *url* is blank.

Return type str/None

harvester.scrappers.utils.**has_param**(*param*)
Generate function, which will check *param* is in html element.

This function can be used as parameter for .find() method in HTMLElement.

harvester.scrappers.utils.**must_contain**(*tag_name*, *tag_content*, *container_tag_name*)
Generate function, which checks if given element contains *tag_name* with string content *tag_content* and also another tag named *container_tag_name*.

This function can be used as parameter for .find() method in HTMLElement.

harvester.scrappers.utils.**content_matchs**(*tag_content*, *content_transformer=None*)
Generate function, which checks whether the content of the tag matchs *tag_content*.

Parameters

- **tag_content** (*str*) – Content of the tag which will be matched thru whole DOM.
- **content_transformer** (*fn, default None*) – Function used to transform all tags before matching.

This function can be used as parameter for .find() method in HTMLElement.

harvester.scrappers.utils.**self_test_idiom**(*fn*)
Perform basic selftest.

Returns When everything is ok.

Return type True

Raises `AssertionError` – When there is some problem.

3.2 Filters

`Filters` are then used to filter data from `Scrappers`, before they are returned. This behavior can be turned off by `USE_DUP_FILTER` and `USE_ALEPH_FILTER` properties of `settings` submodule.

3.2.1 Aleph filter

This module is used to skip Publications, which are already in Aleph.

Note: The module is using fuzzy lookup, see `name_to_vector()` and `compare_names()`.

harvester.filters.aleph_filter.**name_to_vector**(*name*)
Convert *name* to the ASCII vector.

Example

```
>>> name_to_vector("ing. Franta Putšálek")
['putsalek', 'franta', 'ing']
```

Parameters **name** (*str*) – Name which will be vectorized.

Returns Vector created from name.

Return type list

harvester.filters.aleph_filter.**compare_names**(*first*, *second*)
Compare two names in complicated, but more error prone way.

Algorithm is using vector comparison.

Example

```
>>> compare_names("Franta Putšálek", "ing. Franta Putšálek")
100.0
>>> compare_names("F. Putšálek", "ing. Franta Putšálek")
50.0
```

Parameters

- **first** (*str*) – Fisst name as string.
- **second** (*str*) – Second name as string.

Returns Percentage of the similarity.

Return type float

harvester.filters.aleph_filter.**filter_publication**(*publication*, *cmp_authors=True*)
Filter publications based at data from Aleph.

Parameters **publication** (*obj*) – `Publication` instance.

Returns None if the publication was found in Aleph or *publication* if not.

Return type obj/None

3.2.2 Duplication filter

This submodule is used to skip already parsed data.

Each *publication* parameter of the `filter()` is cached and if it is called with same parameter again, None is retuned.

Note: Cache is using simple JSON serialization, so some form of cache persistency is granted. For path to the serialized data, look at `DUP_FILTER_FILE`.

`harvester.filters.dup_filter.`**`save_cache`**(*cache*)
Save cahce to the disk.

Parameters **cache** (*set*) – Set with cached data.

`harvester.filters.dup_filter.`**`load_cache`**()
Load cache from the disk.

Returns Deserialized data from disk.

Return type set

`harvester.filters.dup_filter.`**`filter_publication`**(*publication*, *cache=None*)
Deduplication function, which compares *publication* with samples stored in *cache*. If the match NOT is found, *publication* is returned, else None.

Parameters

- **publication** (*obj*) – `Publication` instance.
- **cache** (*obj*) – Cache which is used for lookups.

Returns Depends whether the object is found in cache or not.

Return type obj/None

3.3 Other parts

There are also other, unrelated parts of this module, which are used to set behavior, or to define representations of the data.

3.3.1 settings

Module is containing all necessary global variables for the package.

Module also has the ability to read user-defined data from two paths:

- `$HOME/_SETTINGS_PATH`
- `/etc/_SETTINGS_PATH`

See `_SETTINGS_PATH` for details.

Note: If the first path is found, other is ignored.

Example of the configuration file (`$HOME/edeposit/harvester.json`):

```
{
    "USE_DUP_FILTER": false,
    "USE_ALEPH_FILTER": false
}
```

Attributes

harvester.settings.**USE_DUP_FILTER = True**
Use duplication filter.

harvester.settings.**USE_ALEPH_FILTER = False**
Use Aleph filter.

harvester.settings.**ALEPH_FILTER_BY_AUTHOR = True**
Consider records from Aleph matching only when the authors are matching?

harvester.settings.**get_all_constants**()
Get list of all uppercase, non-private globals (doesn't start with _).

Returns Uppercase names defined in *globals()* (variables from this module).

Return type list

harvester.settings.**substitute_globals**(*config_dict*)
Set global variables to values defined in *config_dict*.

Parameters **config_dict** (*dict*) – dictionary with data, which are used to set *globals*.

Note: *config_dict* have to be dictionary, or it is ignored. Also all variables, that are not already in globals, or are not types defined in _ALLOWED (str, int, float) or starts with _ are silently ignored.

---

harvester.settings.**DUP_FILTER_FILE = '/home/docs/.edeposit_harvester_cache.json'**
Cache for the deduplicator.

3.3.2 structures

This module contains all structures used in AMQP communication.

class harvester.structures.**Author**(*name*, *URL=None*)
Bases: object

Author name representation.

name
str – String containing author's name.

URL
str – URL to author's profile.

to_namedtuple()
Convert class to namedtuple.

Note: This method is neccessary for AMQP communication.

Returns Representation of the class as simple structure.

Return type namedtuple

class harvester.structures.**Optionals**
Bases: object

Structure for holding optional informations about given publication.

Note: This structure is usually used as container inside Publication.optionals.

sub_title
str, default None – Subtitle of the book.

format
str, default None – Format of the book - A5 for example.

pub_date
str, default None – Date when the book was published.

pub_place
str, default None – Name of the city, where the book was published.

ISBN
str, default None – ISBN of the book.

description
str, default None – Description of the book, which may contain HTML tags and elements!

pages
str, default None – Number of pages.

EAN
str, default None – EAN of the book.

language
str, default None – Language of the book.

edition
str, default None – Edition in which the book was published.

URL
str, default None – URL to the eshop with the book.

binding
str, default None – Binding of the book (*brožovaná* for example).

is_ebook
bool, default False – If True, metadata belongs to ebook.

to_namedtuple()
Convert class to namedtuple.

Note: This method is neccessary for AMQP communication.

Returns Representation of the class as simple structure.

Return type namedtuple

class harvester.structures.**Publication**(*title*, *authors*, *price*, *publisher*)
Bases: object

This class contains only required minimal subset of informations about *publication*.

title
str – Title of the book.

price
str – Price as string with currency.

publisher
str – Publishers name as string.

authors
list – List of `Author` objects. May be blank.

optionals
obj – Reference to `Optionals` object with optional informations.

to_namedtuple()
Convert class and all subclasses (`Author`, `Optionals`) to namedtuple.

Note: This method is neccessary for AMQP communication.

Returns Representation of the class as simple structure.

Return type namedtuple

class harvester.structures.**Publications**
Bases: `harvester.structures.Publication`

AMQP communication structured used to hold the transfered informations.

publications
list – List of `Publication` namedtuples.

3.3.3 Autogenerator

Last submodule is `Autoparser`, which makes creating new parsers easier.

conf_reader

Functions which allows to read serialized informations for autoparser.

harvester.autoparser.conf_reader.**read_config**(*file_name*)
Read YAML file with configuration and pointers to example data.

Parameters **file_name** (*str*) – Name of the file, where the configuration is stored.

Returns Parsed and processed data (see `_process_config_item()`).

Return type dict

Example YAML file:: html: simple_xml.xml first:

data: i wan't this required: true notfoundmsg: Can't find variable $name.

second: data: and this

— html: simple_xml2.xml first:

data: something wanted required: true notfoundmsg: Can't find variable $name.

second: data: another wanted thing

vectors

This module contains functions to convert *DOM* relations to *path-like* lists of elements defined by tag names and parameters.

harvester.autoparser.vectors.**el_to_path_vector**(*el*)
Convert *el* to vector of foregoing elements.

Attr: el (obj): Double-linked HTMLElement instance.

Returns HTMLElements which considered as path from root to *el*.

Return type list

harvester.autoparser.vectors.**common_vector_root**(*vec1*, *vec2*)
Return common root of the two vectors.

Parameters

- **vec1** (*list/tuple*) – First vector.
- **vec2** (*list/tuple*) – Second vector.

Usage example:

```
>>> common_vector_root([1, 2, 3, 4, 5], [1, 2, 8, 9, 0])
[1, 2]
```

Returns Common part of two vectors or blank list.

Return type list

harvester.autoparser.vectors.**find_common_root**(*elements*)
Find root which is common for all *elements*.

Parameters **elements** (*list*) – List of double-linked HTMLElement objects.

Returns Vector of HTMLElement containing path to common root.

Return type list

path_patterns

This module defines path-constructor functions and containers for data.

Containers are later used for validation of the paths in other examples and for generator, which creates the parser.

class harvester.autoparser.path_patterns.**NeighCall**(*tag_name*, *params*, *fn_params*)
Class used to store informations about neighbour calls, generated by `_neighbour_to_path_call()`.

tag_name
str – Name of the container for the data.

params
dict – Parameters for the fontainer.

fn_params
list – Parameters for the fuction which will find neighbour (see `has_neigh()`).

class harvester.autoparser.path_patterns.**PathCall**(*call_type*, *index*, *params*)
Container used to hold data, which will be used as parameter to call search functions in *DOM*.

Parameters

- **call_type** (*str*) – Determines type of the call to the HTMLElement method.
- **index** (*int*) – Index of the item after *call_type* function is called.
- **params** (*dict*) – Another parameters for *call_type* function.

class harvester.autoparser.path_patterns.**Chained**(*chain*)
Container to hold parameters of the chained calls.

Parameters **chain** (*list*) – List of `PathCall` classes.

call_type
Property added to make sure, that `Chained` is interchangeable with `PathCall`.

harvester.autoparser.path_patterns.**neighbours_pattern**(*element*)
Look for negihbours of the *element*, return proper `PathCall`.

Parameters **element** (*obj*) – HTMLElement instance of the object you are looking for.

Returns List of `PathCall` instances.

Return type list

harvester.autoparser.path_patterns.**predecesors_pattern**(*element*, *root*)
Look for *element* by its predecesors.

Parameters

- **element** (*obj*) – HTMLElement instance of the object you are looking for.
- **root** (*obj*) – Root of the *DOM*.

Returns `[PathCall()]` - list with one `PathCall` object (to allow use with `.extend(predecesors_pattern())`).

Return type list

utils

This module contains number of functions, which are used at multiple places in autoparser.

harvester.autoparser.utils.**handle_encodnig**(*html*)
Look for encoding in given *html*. Try to convert *html* to utf-8.

Parameters **html** (*str*) – HTML code as string.

Returns HTML code encoded in UTF.

Return type str

harvester.autoparser.utils.**content_matchs**(*tag_content*, *content_transformer=None*)
Generate function, which checks whether the content of the tag matchs *tag_content*.

Parameters

- **tag_content** (*str*) – Content of the tag which will be matched thru whole DOM.
- **content_transformer** (*fn, default None*) – Function used to transform all tags before matching.

Returns True for every matching tag.

Return type bool

Note: This function can be used as parameter for `.find()` method in HTMLElement.

harvester.autoparser.utils.**is_equal_tag**(*element*, *tag_name*, *params*, *content*)
Check is *element* object match rest of the parameters.

All checks are performed only if proper attribute is set in the HTMLElement.

Parameters

- **element** (*obj*) – HTMLElement instance.
- **tag_name** (*str*) – Tag name.
- **params** (*dict*) – Parameters of the tag.
- **content** (*str*) – Content of the tag.

Returns True if everyhing matchs, False otherwise.

Return type bool

harvester.autoparser.utils.**has_neigh**(*tag_name*, *params=None*, *content=None*, *left=True*)
This function generates functions, which matches all tags with neighbours defined by parameters.

Parameters

- **tag_name** (*str*) – Tag has to have neighbour with this tagname.
- **params** (*str*) – Tag has to have neighbour with this parameters.
- **params** – Tag has to have neighbour with this content.
- **left** (*bool, default True*) – Tag has to have neigbour on the left, or right (set to `False`).

Returns True for every matching tag.

Return type bool

Note: This function can be used as parameter for `.find()` method in HTMLElement.

generator

This module contains number of template generators, which generates all the python code for the parser.

harvester.autoparser.generator.**IND = ' '**
Indentation.

harvester.autoparser.generator.**_index_idiom**(*el_name*, *index*, *alt=None*)
Generate string where *el_name* is indexed by *index* if there are enough items or *alt* is returned.

Parameters

- **el_name** (*str*) – Name of the *container* which is indexed.
- **index** (*int*) – Index of the item you want to obtain from container.
- **alt** (*whatever, default None*) – Alternative value.

Returns Python code.

Return type str

Live example::

```
>>> import generator as g
>>> print g._index_idiom("xex", 0)
    # pick element from list
    xex = xex[0] if xex else None
>>> print g._index_idiom("xex", 1, "something")
# pick element from list
xex = xex[1] if len(xex) - 1 >= 1 else 'something'
```

harvester.autoparser.generator.**_required_idiom**(*tag_name*, *index*, *notfoundmsg*)

Generate code, which make sure that *tag_name* has enoug items.

Parameters

- **tag_name** (*str*) – Name of the container.
- **index** (*int*) – Index of the item you want to obtain from container.
- **notfoundmsg** (*str*) – Raise `UserWarning` with debug data and following message.

Returns Python code.

Return type str

harvester.autoparser.generator.**_find_template**(*parameters*, *index*, *required=False*, *notfoundmsg=None*)

Generate `.find()` call for HTMLElement.

Parameters

- **parameters** (*list*) – List of parameters for `.find()`.
- **index** (*int*) – Index of the item you want to get from `.find()` call.
- **required** (*bool, default False*) – Use `_required_idiom()` to returned data.
- **notfoundmsg** (*str, default None*) – Message which will be used for `_required_idiom()` if the item is not found.

Returns Python code.

Return type str

Live example::

```
>>> print g._find_template(["<xex>"], 3)
    el = dom.find('<xex>')
    # pick element from list
    el = el[3] if len(el) - 1 >= 3 else None
```

harvester.autoparser.generator.**_wfind_template**(*use_dom*, *parameters*, *index*, *required=False*, *notfoundmsg=None*)

Generate `.wfind()` call for HTMLElement.

Parameters

- **use_dom** (*bool*) – Use `dom` as tag name. If `False`, `el` is used.
- **parameters** (*list*) – List of parameters for `.wfind()`.
- **index** (*int*) – Index of the item you want to get from `.wfind()` call.
- **required** (*bool, default False*) – Use `_required_idiom()` to returned data.

- **notfoundmsg** (*str, default None*) – Message which will be used for `_required_idiom()` if the item is not found.

Returns Python code.

Return type str

Live example::

```
>>> print g._wfind_template(True, ["<xex>"], 3)
    el = dom.wfind('<xex>').childs
    # pick element from list
    el = el[3] if len(el) - 1 >= 3 else None
```

harvester.autoparser.generator.**_match_template**(*parameters*, *index*, *required=False*, *notfoundmsg=None*)

Generate `.match()` call for HTMLElement.

Parameters

- **parameters** (*list*) – List of parameters for `.match()`.
- **index** (*int*) – Index of the item you want to get from `.match()` call.
- **required** (*bool, default False*) – Use `_required_idiom()` to returned data.
- **notfoundmsg** (*str, default None*) – Message which will be used for `_required_idiom()` if the item is not found.

Returns Python code.

Return type str

Live example::

```
>>> print g._match_template(["<xex>"], 3)
    el = dom.match('<xex>')
    # pick element from list
    el = el[3] if len(el) - 1 >= 3 else None
```

harvester.autoparser.generator.**_neigh_template**(*parameters*, *index*, *left=True*, *required=False*, *notfoundmsg=None*)

Generate neighbour matching call for HTMLElement, which returns only elements with required neighbours.

Parameters

- **parameters** (*list*) – List of parameters for `.match()`.
- **index** (*int*) – Index of the item you want to get from `.match()` call.
- **left** (*bool, default True*) – Look for neigbour in the left side of el.
- **required** (*bool, default False*) – Use `_required_idiom()` to returned data.
- **notfoundmsg** (*str, default None*) – Message which will be used for `_required_idiom()` if the item is not found.

Returns Python code.

Return type str

harvester.autoparser.generator.**_get_parser_name**(*var_name*)

Parser name composer.

Parameters **var_name** (*str*) – Name of the variable.

Returns Parser function name.

Return type str

harvester.autoparser.generator.**_generate_parser**(*name*, *path*, *required=False*, *notfoundmsg=None*)

Generate parser named *name* for given *path*.

Parameters

- **name** (*str*) – Basename for the parsing function (see `_get_parser_name()` for details).
- **path** (*obj*) – `PathCall` or `Chained` instance.
- **required** (*bool, default False*) – Use `_required_idiom()` to returned data.
- **notfoundmsg** (*str, default None*) – Message which will be used for `_required_idiom()` if the item is not found.

Returns Python code for parsing *path*.

Return type str

harvester.autoparser.generator.**_unittest_template**(*config*)

Generate unittests for all of the generated code.

Parameters **config** (*dict*) – Original configuration dictionary. See `conf_reader` for details.

Returns Python code.

Return type str

harvester.autoparser.generator.**generate_parsers**(*config*, *paths*)

Generate parser for all *paths*.

Parameters

- **config** (*dict*) – Original configuration dictionary used to get matches for unittests. See `conf_reader` for details.
- **paths** (*dict*) – Output from `select_best_paths()`.

Returns Python code containing all parsers for *paths*.

Return type str

CHAPTER 4

AMQP connection

AMQP communication is handled by the edeposit.amqp module, specifically by the `edeposit_amqp_harvester.py` script.

CHAPTER 5

Source code

This project is released as opensource (GPL) and source codes can be found at GitHub:

- https://github.com/edeposit/edeposit.amqp.harvester

5.1 Installation

Module is hosted at PYPI, and can be easily installed using PIP:

```
sudo pip install edeposit.amqp.harvester
```

CHAPTER 6

Testing

Almost every feature of the project is tested in unit/integration tests. You can run this tests using provided `run_tests.sh` script, which can be found in the root of the project.

6.1 Requirements

This script expects that pytest is installed. In case you don't have it yet, it can be easily installed using following command:

```
pip install --user pytest
```

or for all users:

```
sudo pip install pytest
```

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*

e

h

Symbols

A

B

C

D

E

F

G

H

I