ChemDataExtractor Documentation

Release v2.0.0

University of Cambridge, Molecular Engineering Group

Aug 23, 2021

Contents:

1 Getting Started **3**
- 1.1 The ChemDataExtractor Toolkit . . . 4
- 1.2 Installation . . . 4
- 1.3 Reading a Document . . . 6
- 1.4 Advanced Topics . . . 8
- 1.5 Contributing . . . 22

2 Examples **25**
- 2.1 Creating new units and dimensions . . . 25

3 API Documentation **29**
- 3.1 chemdataextractor . . . 29
- 3.2 .biblio . . . 31
- 3.3 .cli . . . 34
- 3.4 .doc . . . 37
- 3.5 .eval . . . 56
- 3.6 .model . . . 56
- 3.7 .nlp . . . 79
- 3.8 .parse . . . 91
- 3.9 .reader . . . 104
- 3.10 .relex . . . 111
- 3.11 .scrape . . . 115
- 3.12 .text . . . 140

4 v2.0 Migration Guide **149**
- 4.1 Overview . . . 149
- 4.2 Changes to ChemDataExtractor . . . 149
- 4.3 Migrating Existing Code . . . 154
- 4.4 Upgrading Existing Code . . . 155

5 Change Log **161**
- 5.1 v2.0 (2019-09-xx) . . . 161
- 5.2 v1.3.0 (2017-02-03) . . . 161
- 5.3 v1.2.3 (2017-01-22) . . . 162
- 5.4 v1.2.2 (2016-11-02) . . . 162
- 5.5 v1.2.1 (2016-10-24) . . . 162
- 5.6 v1.2.0 (2016-10-11) . . . 162

5.7 v1.1.1 (2016-10-04) . . . 162
5.8 v1.1.0 (2016-10-03) . . . 163

6 License/Citing **165**
6.1 ChemDataExtractor v2 is released under the MIT license. . . . 165
6.2 ChemDataExtractor License . . . 166

7 Indices and tables **167**

Python Module Index **169**

Index **171**

ChemDataExtractor is a toolkit for extracting chemical information from the scientific literature. Check out the Online Demo!

Features:

- HTML, XML and PDF document readers
- Chemistry-aware natural language processing pipeline
- Chemical named entity recognition
- Rule-based parsing grammars for property and spectra extraction
- Table parser for extracting tabulated data
- Document processing to resolve data interdependencies

CHAPTER 1

Getting Started

Contents

- *Getting Started*
  - *The ChemDataExtractor Toolkit*
  - *Installation*
  - *Reading a Document*
  - *Advanced Topics*
    - *Document Readers*
    - *Scraping Structured Data*
    - *Cleaners*
    - *Chemical Records*
    - *Tokenization*
    - *Part-of-speech Tagging*
    - *Lexicon*
    - *Abbreviation Detection*
    - *Command Line Interface*
    - *Regular Expressions*
    - *Creating a New Property Parser*
    - *ChemDataExtractor REST API*
    - *"Where do I find. . . ?"*
  - *Contributing*

* *Documenting Code*

1.1 The ChemDataExtractor Toolkit

The ChemDataExtractor toolkit is an advanced natural language processing pipeline for extracting chemical property information from the scientific literature. The theory behind the toolkit can be found in the following papers:

Original paper outlining the CDE workflow:

Swain, M. C., & Cole, J. M. "ChemDataExtractor: A Toolkit for Automated Extraction of Chemical Information from the Scientific Literature", J. Chem. Inf. Model. 2016, 56 (10), pp 1894–1904 10.1021/acs.jcim.6b00207.

Paper describing the Snowball algorithm:

Court, C. J., & Cole, J. M. (2018). Auto-generated materials database of Curie and Néel temperatures via semi-supervised relationship extraction. Scientific data, 5, 180111. 10.1038/sdata.2018.111

Paper describing the enhancements made in CDE 2.0 and TableDataExtractor:

TO BE ADDED

and the associated website https://chemdataextractor.org.

The general process for extracting information from scientific text is as follows:

1. Break a document down into its consituent elements (Title, Paragraphs, Sentences, Tables, Figures. . .)
2. Tokenize the text to isolate individual tokens
3. Apply Part-of-speech tagging to identify the semantic role of each token
4. Detect Chemical Named Entites using machine learning
5. Parse text and tables with nested rules to identify chemical relationships
6. Resolve the interdependencies between the different elements
7. Output a set of mutually consistent chemical records

This pipeline enables ChemDataExtractor to extract chemical information in a completely domain-independent manner.

1.2 Installation

To get up and running with ChemDataExtractor, you will need to install the python toolkit and then download the necessary data files. There are a few different ways to download and install the ChemDataExtractor toolkit.

Note: A Python version earlier than 3.8 must be used with ChemDataExtractor.

Option 1. Using conda

This method is recommended for all Windows users, as well as beginners on all platforms who don't already have Python installed. Anaconda Python is a self-contained Python environment that is particularly useful for scientific applications.

Start by installing Miniconda, which includes a complete Python distribution and the conda package manager, or Anaconda, which additionally includes many pre-installed packages, including NumPy and Matplotlib. Choose the Python 3.5 version, unless you have a particular reason why you must use Python 2.7.

Once installed, at the command line, run:

```
$ conda install -c chemdataextractor chemdataextractor
```

This command installs the chemdataextractor package from the chemdataextractor conda channel.

Option 2. Using pip

If you already have Python installed, it's easiest to install the ChemDataExtractor package using pip. At the command line, run:

```
$ pip install ChemDataExtractor
```

On Windows, this will require the Microsoft Visual C++ Build Tools to be installed. If you don't already have pip installed, you can install it using `get-pip.py`.

Option 3. Download the Latest Release

Alternatively, download the latest release manually from github or the ChemDataExtractor website and install it yourself by running:

```
$ cd chemdataextractor
$ python setup.py install
```

The `setup.py` command will install ChemDataExtractor in your site-packages folder so it is automatically available to all your python scripts.

You can also get the latest release by cloning the git source code repository from https://github.com/CambridgeMolecularEngineering/chemdataextractor/.

Getting the Data Files

In order to function, ChemDataExtractor requires a variety of data files, such as machine learning models, dictionaries, and word clusters. Get these by running:

```
$ cde data download
```

This will download all the necessary data files to the data directory. Run:

```
$ cde data where
```

to see where this is.

Updating

Upgrade your installation to the latest version at any time using conda or pip, matching the method you used originally to install it. For conda, run:

```
$ conda update -c chemdataextractor chemdataextractor
```

For pip, run:

```
$ pip install --upgrade ChemDataExtractor
```

Either way, always remember to download any new data files after doing this:

```
$ cde data download
```

1.3 Reading a Document

Most commonly, you want to pass an entire document file to ChemDataExtractor. ChemDataExtractor comes with a number of built-in Document readers that can read HTML, PDF and XML files. These readers are responsible for detecting the different elements of a document and recompiling them into a single consistent document structure:

```
>>> from chemdataextractor import Document
>>> f = open('paper.html', 'rb')
>>> doc = Document.from_file(f)
```

Each reader will be tried in turn until one is successfully able to read the file. If you know exactly which readers you want to use, it is possible to specify a list as an optional parameter:

```
>>> f = open('rsc_article.html', 'rb')
>>> doc = Document.from_file(f, readers=[RscHtmlReader()])
```

Note: Always open files in binary mode by using the 'rb' parameter.

Alternatively, you can load a document into ChemDataExtractor by passing it some text:

```
>>> doc = Document('UV-vis spectrum of 5,10,15,20-Tetra(4-carboxyphenyl)porphyrin in␣
↪Tetrahydrofuran (THF).')
```

At present, the available readers are:

- AcsHtmlReader - For ACS HTML articles
- RscHtmlReader - For RSC HTML articles
- NlmXmlReader - For NLM/JATS XML (e.g. from PubMed Central)
- UsptoXmlReader - For patent XML from the US Patent Office
- CsspHtmlReader - For ChemSpider SyntheticPages
- XmlReader - Generic XML
- HtmlReader - Generic HTML
- PdfReader - Generic PDF
- PlainTextReader - Generic plain text

The HTML and XML readers can determine document structure such as headings, paragraphs, and tables with high accuracy. However, this is much harder to achieve with the PDF and plain text readers.

More information about document readers can be found in the *Advanced Topics*.

Document Elements

Once read, documents are represented by a single linear stream of *element* objects. This stream is now independent of the initial document type or the source:

```
>>> doc.elements
[Title('A very important scientific article'),
Heading('Abstract'),
Paragraph('The first paragraph of text...'),
...]
```

Element types include Title, Heading, Paragraph, Citation, Table, Figure, Caption and Footnote. You can retrieve a specific element by its index within the document:

```
>>> para = doc.elements[14]
>>> para
Paragraph('1,4-Dibromoanthracene was prepared from 1,4-diaminoanthraquinone. 1H NMR␣
↪spectra were recorded on a 300 MHz BRUKER DPX300 spectrometer.')
```

You can also get the individual sentences of a paragraph:

```
>>> para.sentences
[Sentence('1,4-Dibromoanthracene was prepared from 1,4-diaminoanthraquinone.', 0, 65),
Sentence('1H NMR spectra were recorded on a 300 MHz BRUKER DPX300 spectrometer.', 66,␣
↪135)]
```

Or, the individual tokens:

```
>>> para.tokens
[[Token('1,4-Dibromoanthracene', 0, 21),
Token('was', 22, 25),
Token('prepared', 26, 34),
Token('from', 35, 39),
Token('1,4-diaminoanthraquinone', 40, 64),
Token('.', 64, 65)],
[Token('1H', 66, 68),
Token('NMR', 69, 72),
Token('spectra', 73, 80),
Token('were', 81, 85),
Token('recorded', 86, 94),
Token('on', 95, 97),
Token('a', 98, 99),
Token('300', 100, 103),
Token('MHz', 104, 107),
Token('BRUKER', 108, 114),
Token('DPX300', 115, 121),
Token('spectrometer', 122, 134),
Token('.', 134, 135)]]
```

as well as a list of individual chemical entity mentions (CEMs) of the document:

```
>>> doc.cems
[Span('5,10,15,20-Tetra(4-carboxyphenyl)porphyrin', 19, 61),
 Span('THF', 82, 85),
 Span('Tetrahydrofuran', 65, 80)]
```

Each mention is returned as a `Span`, which contains the mention text, as well as the start and end character offsets within the containing document element.

You can also output the abbreviations found in the document:

```
>>> doc.abbreviation_definitions
[([u'THF'], [u'Tetrahydrofuran'], u'CM')]
```

The *records* method, combines all the chemical mentions, abbreviations and properties found each chemical entity (see *Examples*):

```
>>> doc.records
[<Compound>, <Compound>]
>>> doc.records[0].serialize()
{'names': ['5,10,15,20-Tetra(4-carboxyphenyl)porphyrin']}
>>> doc.records[1].serialize()
{'names': ['Tetrahydrofuran', 'THF']}
```

Which file formats are best?

While ChemDataExtractor supports documents in a wide variety of formats, some are better suited for extraction than others. If there is an HTML or XML version available, that is normally the best choice.

Wherever possible, avoid using the PDF version of a paper or patent. At best, the text will be interpretable, but it is extremely difficult to reliably distinguish between headings, captions and main body text. At worst, the document will just consist of a scanned image of each page, and it won't be possible to extract any of the text at all. You can get some idea of what ChemDataExtractor can see in a PDF by looking at the result of copying-and-pasting from the document.

For scientific articles, most publishers offer a HTML version alongside the PDF version. Normally, this will open as a page in your web browser. Just choose "Save As..." and ensure the selected format is "HTML" or "Page Source" to save a copy of the HTML file to your computer.

Most patent offices provide XML versions of their patent documents, but these can be hard to find. Two useful resources are the USPTO Bulk Data Download Service and the EPO Open Patent Services API.

More information

The *Advanced Topics* section provides more detailed instructions for advanced ChemDataExtractor functionality.

1.4 Advanced Topics

1.4.1 Document Readers

The document readers present in the `chemdataextractor.reader` package are a set of tools for identifying the elements of scientific documents. The HTML and XML from each publisher is slightly different, meaning we once again need multiple different readers. New users are often confused about the structure of these readers, and so this section attempts to explain their functionality more clearly.

As an example, lets look at the `chemdataextractor.reader.rsc.RscHtmlReader` class:

```
class RscHtmlReader(HtmlReader):
    """Reader for HTML documents from the RSC."""

    cleaners = [clean, replace_rsc_img_chars, space_references]

    root_css = '#wrapper, html'
```

(continues on next page)

(continued from previous page)

```
    title_css = 'h1, .title_heading'
    heading_css = 'h2, h3, h4, h5, h6, .a_heading, .b_heading, .c_heading, .c_heading_
↪indent, .d_heading, .d_heading_indent'
    citation_css = 'span[id^="cit"]'
    table_css = 'div[class="rtable__wrapper"]'
    table_caption_css = 'div[class="table_caption"]'
    table_id_css = 'span[id^="tab"]::attr("id")'
    table_head_row_css = 'thead'
    table_body_row_css = 'tr'
    table_footnote_css = '.table_caption + table tfoot tr th .sup_inf'
    reference_css = 'small sup a, a[href^="#cit"], a[href^="#fn"], a[href^="#tab"]'
    figure_css = '.image_table'
    figure_caption_css = '.graphic_title'
    figure_label_css = 'span[id^="fig"]::attr("id")'
    ignore_css = '.table_caption + table, .left_head, sup span.sup_ref, ' \
                 'a[href^="#fn"], .PMedLink, p[class="header_text"], ' \
                 'a[href^="#tab"], span[class="sup_ref"]'
```

As you can see, we have a number of CSS Selectors that are used to select particular elements from an RSC HTML document. Here, the variable names are important, and must follow the format `<element_name>_css`, as this tells the `BaseReader` what to name the selected element.

These elements are found by examining the HTML. For example, if you find a paper from the RSC web pages, open the HTML version, then right-click and chose "view page source" you will be able to see the raw HTML. If you are unfamiliar with HTML and CSS I recommend going through the TutorialsPoint HTML tutorial and CSS tutorial.

It should also be mentioned that these Readers override the element variables from the base `HTMLReader` class. Similarly, if you want to analyse an XML document, you should override from the `XMLReader` class. I first recommend using the base readers, to see how they perform, then write a new reader if you have to.

1.4.2 Scraping Structured Data

ChemDataExtractor contains a `scrape` package for extracting structured information from HTML and XML files. This is most useful for obtaining bibliographic data, but can be used for any kind of data that has been marked up with HTML or XML tags in source documents.

Included Scrapers

ChemDataExtractor comes with ready-made scraping tools for web pages on the RSC and ACS web sites, as wells as for XML files in the NLM JATS format as used by PubMed Central and others:

```
>>> from chemdataextractor.scrape import Selector
>>> from chemdataextractor.scrape.pub.rsc import RscHtmlDocument
>>>
>>> htmlstring = open('rsc_example.html').read()
>>> sel = Selector.from_text(htmlstring)
>>> scrape = RscHtmlDocument(sel)
>>> print(scrape.publisher)
'Royal Society of Chemistry'
>>> scrape.serialize()
{'publisher': 'Royal Society of Chemistry', 'language': 'en', 'title': 'The Title'}
```

Custom Scrapers

As an example, here is a very simple HTML file that we want to scrape some data from:

```
<html>
  <head>
    <title>Example document</title>
    <meta name="citation_publication_date" content="2016-10-03">
  </head>
  <body>
    <p class="abstract">Abstract goes here...</p>
    <p class="para">Another paragraph here...</p>
  </body>
</html>
```

Defining an Entity

To use the `scrape` package, we define an `Entity` that contains `Fields` that describe how to extract the desired content in a declarative fashion:

```
from chemdataextractor.scrape import Entity

class ExampleDocument(Entity):
    title = StringField('title')
    abstract = StringField('.abstract')
    date_published = DateTimeField('meta[name="citation_publication_date"]::attr(
↪"content")')
```

Each field uses a CSS selector to describe where to find the data in the document.

XPath Expressions

It is possible to use XPath expressions instead of CSS selectors, if desired. Just add the parameter `xpath=True` to the field arguments:

```
date_published = DateTimeField('//meta[@name="citation_publication_date"]/@content',␣
↪xpath=True)
```

Processors

Processors perform transformations on the extracted text.

The Selector

The `Selector` is inspired by the Scrapy text mining tool. It provides a convenient unified interface for 'selecting' parts of XML and HTML documents for extraction. `Entity` classes make use of it behind the scenes, but for simple cases, it can be quicker and easier to use it directly to extract information.

Create a selector from a file:

```
>>> htmlstring = open('rsc_example.html').read()
>>> sel = Selector.from_text(htmlstring)
```

Now, instead of passing the selector to an Entity, you can query it with CSS:

```
>>> sel.css('head')
```

This returns a `SelectorList`, meaning you can chain queries. Call `extract()` or `extract_first()` on the returned `SelectorList` to get the extracted content:

```
>>> sel.css('head').css('title').extract_first()
'Example document'
>>> sel.css('p')
['Abstract goes here...', 'Another paragraph here...']
```

1.4.3 Cleaners

You will see in the above code that we have specified a number of cleaners. Cleaners attempt to fix systematic formatting errors in the HTML/XML. A classic problem is spacing around references. For example some HTML may look like:

```
<div>
    <p>This is a result that was retrieved from
        <a><sup><span class=sup_ref>[1]</span><sup></a>.
    </p>
</div>
```

When parsing, ChemDataExtractor will output:

```
Paragraph(text='This a result that was retrieved from[1].',...)
```

So we need a cleaner whose job is to put a space between text and references. In the `RscHtmlReader` class we specify a list of cleaners to act on the text:

```
cleaners = [clean, replace_rsc_img_chars, space_references]
```

and the corresponding `space_references` cleaner looks like:

```
def space_references(document):
    """Ensure a space around reference links, so there's a gap when they are removed."
↪""
    for ref in document.xpath('.//a/sup/span[@class="sup_ref"]'):
        a = ref.getparent().getparent()
        if a is not None:
            atail = a.tail or ''
            if not atail.startswith(')') and not atail.startswith(',') and not atail.
↪startswith(' '):
                a.tail = ' ' + atail
    return document
```

Note that we don't explicitly need to call the cleaner as this is handled by the `BaseReader` class.

1.4.4 Chemical Records

ChemDataExtractor processes each document element separately to extract the chemical information, and then merges data together from every element in the document to produce a single record for each unique chemical entity.

Consider this simple document as an example:

```
>>> from chemdataextractor.doc import Document, Heading, Paragraph
>>> doc = Document(
    Heading('5,10,15,20-Tetra(4-carboxyphenyl)porphyrin (3).'),
    Paragraph('m.p. 90°C.'),
    Paragraph('Melting points were measured in Tetrahydrofuran (THF).'),
    )
```

Get the records for each element using the records property:

```
>>> doc[0].records.serialize()
[{'Compound': {'names': ['5,10,15,20-Tetra(4-carboxyphenyl)porphyrin'], 'labels': ['3
↪']}}]
>>> doc[1].records.serialize()
[{'MeltingPoint': {'raw_value': '90', 'raw_units': '°C', 'value': [90.0], 'units':
↪'Celsius^(1.0)'}}]
>>> doc[2].records.serialize()
[{'Compound': {'names': ['THF', 'Tetrahydrofuran']}}, {'Compound': {'names': ['THF',
↪'Tetrahydrofuran']}}]
```

Due to the data interdependencies between the different document elements, each isn't so useful individually. Instead, it's normally much more useful to get the combined records for the entire document:

```
>>> doc.records.serialize()
[{'Compound': {'names': ['5,10,15,20-Tetra(4-carboxyphenyl)porphyrin'], 'labels': ['3
↪']}},
 {'Compound': {'names': ['THF', 'Tetrahydrofuran']}},
 {'MeltingPoint': {'raw_value': '90',
                   'raw_units': '°C',
                   'value': [90.0],
                   'units': 'Celsius^(1.0)',
                   'compound': {'Compound': {'names': ['5,10,15,20-Tetra(4-
↪carboxyphenyl)porphyrin'],
                                             'labels': ['3']}}}}]
```

ChemDataExtractor has merged the information from all the elements into two unique chemical records.

1.4.5 Tokenization

Sentence Tokenization

Use the `sentences` property on a text-based document element to perform sentence segmentation:

```
>>> from chemdataextractor.doc import Paragraph
>>> para = Paragraph('1,4-Dibromoanthracene was prepared from 1,4-
↪diaminoanthraquinone. 1H NMR spectra were recorded on a 300 MHz BRUKER DPX300␣
↪spectrometer.')
>>> para.sentences
[Sentence('1,4-Dibromoanthracene was prepared from 1,4-diaminoanthraquinone.', 0, 65),
 Sentence('1H NMR spectra were recorded on a 300 MHz BRUKER DPX300 spectrometer.', 66,
↪ 135)]
```

Each sentence object is a document element in itself, and additionally contains the start and end character offsets within it's parent element.

Word Tokenization

Use the `tokens` property to get the word tokens:

```
>>> para.tokens
[[Token('1,4-Dibromoanthracene', 0, 21),
  Token('was', 22, 25),
  Token('prepared', 26, 34),
```

(continues on next page)

(continued from previous page)

```
  Token('from', 35, 39),
  Token('1,4-diaminoanthraquinone', 40, 64),
  Token('.', 64, 65)],
 [Token('1H', 66, 68),
  Token('NMR', 69, 72),
  Token('spectra', 73, 80),
  Token('were', 81, 85),
  Token('recorded', 86, 94),
  Token('on', 95, 97),
  Token('a', 98, 99),
  Token('300', 100, 103),
  Token('MHz', 104, 107),
  Token('BRUKER', 108, 114),
  Token('DPX300', 115, 121),
  Token('spectrometer', 122, 134),
  Token('.', 134, 135)]]
```

This also works on an individual sentence:

```
>>> para.sentences[0].tokens
[Token('1,4-Dibromoanthracene', 0, 21),
 Token('was', 22, 25),
 Token('prepared', 26, 34),
 Token('from', 35, 39),
 Token('1,4-diaminoanthraquinone', 40, 64),
 Token('.', 64, 65)]
```

There are also `raw_sentences` and `raw_tokens` properties that return strings instead of `Sentence` and `Token` objects.

Using Tokenizers Directly

All tokenizers have a `tokenize` method that takes a text string and returns a list of tokens:

```
>>> from chemdataextractor.nlp.tokenize import ChemWordTokenizer
>>> cwt = ChemWordTokenizer()
>>> cwt.tokenize('1H NMR spectra were recorded on a 300 MHz BRUKER DPX300␣
↪spectrometer.')
['1H', 'NMR', 'spectra', 'were', 'recorded', 'on', 'a', '300', 'MHz', 'BRUKER',
↪'DPX300', 'spectrometer', '.']
```

There is also a `span_tokenize` method that returns the start and end offsets of the tokens in terms of the characters in the original string:

```
>>> cwt.span_tokenize('1H NMR spectra were recorded on a 300 MHz BRUKER DPX300␣
↪spectrometer.')
[(0, 2), (3, 6), (7, 14), (15, 19), (20, 28), (29, 31), (32, 33), (34, 37), (38, 41),␣
↪(42, 48), (49, 55), (56, 68), (68, 69)]
```

1.4.6 Part-of-speech Tagging

ChemDataExtractor contains a chemistry-aware Part-of-speech tagger. Use the `pos_tagged_tokens` property on a document element to get the tagged tokens:

```
>>> s = Sentence('1H NMR spectra were recorded on a 300 MHz BRUKER DPX300␣
↪spectrometer.')
>>> s.pos_tagged_tokens
[('1H', 'NN'),
 ('NMR', 'NN'),
 ('spectra', 'NNS'),
 ('were', 'VBD'),
 ('recorded', 'VBN'),
 ('on', 'IN'),
 ('a', 'DT'),
 ('300', 'CD'),
 ('MHz', 'NNP'),
 ('BRUKER', 'NNP'),
 ('DPX300', 'NNP'),
 ('spectrometer', 'NN'),
 ('.', '.')]
```

Using Taggers Directly

All taggers have a `tag` method that takes a list of token strings and returns a list of (token, tag) tuples:

```
>>> from chemdataextractor.nlp.pos import ChemCrfPosTagger
>>> cpt = ChemCrfPosTagger()
>>> cpt.tag(['1H', 'NMR', 'spectra', 'were', 'recorded', 'on', 'a', '300', 'MHz',
↪'BRUKER', 'DPX300', 'spectrometer', '.'])
[('1H', 'NN'),
 ('NMR', 'NN'),
 ('spectra', 'NNS'),
 ('were', 'VBD'),
 ('recorded', 'VBN'),
 ('on', 'IN'),
 ('a', 'DT'),
 ('300', 'CD'),
 ('MHz', 'NNP'),
 ('BRUKER', 'NNP'),
 ('DPX300', 'NNP'),
 ('spectrometer', 'NN'),
 ('.', '.')]
```

1.4.7 Lexicon

As ChemDataExtractor processes documents, it adds each unique word that it encounters to the `Lexicon` as a `Lexeme`. Each `Lexeme` stores various word features, so they don't have to be re-calculated for every occurrence of that word.

You can access the Lexeme for a token using the `lex` property:

```
>>> s = Sentence('Sulphur and Oxygen.')
>>> s.tokens[0]
Token('Sulphur', 0, 7)
>>> s.tokens[0].lex.normalized
'sulfur'
>>> s.tokens[0].lex.is_hyphenated
False
```

(continues on next page)

(continued from previous page)

```
>>> s.tokens[0].lex.cluster
'11011101100110'
```

1.4.8 Abbreviation Detection

Abbreviation detection is done using a method based on the algorithm in Schwartz & Hearst 2003:

```
>>> p = Paragraph(u'Dye-sensitized solar cells (DSSCs) with ZnTPP = Zinc␣
↪tetraphenylporphyrin.')
>>> p.abbreviation_definitions
[([u'ZnTPP'], [u'Zinc', u'tetraphenylporphyrin'], u'CM'),
 ([u'DSSCs'], [u'Dye', u'-', u'sensitized', u'solar', u'cells'], None)]
```

Abbreviation definitions are returned as tuples containing the abbreviation, the long name, and an entity tag. The entity tag is `CM` if the abbreviation is for a chemical entity, otherwise it is `None`.

1.4.9 Command Line Interface

ChemDataExtractor includes a command line tool that can be accessed by typing `cde` at a command prompt.

Using the Command Line

On a Mac, open the **Terminal** app, which you can find by searching or by looking in the **Utilities** folder in the **Applications** folder.

On Windows, use the **Command Prompt** or **PowerShell**.

For each of the commands below, type or paste the command, then press **Return** to run it.

For any command, add `--help` to the end to get information on how to use it.

Downloading Data Files

In order to function, ChemDataExtractor requires a variety of data files, such as machine learning models, dictionaries, and word clusters.

Data commands:

- `cde data download`: Download data files.
- `cde data clean`: Prune data that is no longer required.
- `cde data list`: List active data packages.
- `cde data where`: Print path to data directory.

Extracting Data

To run ChemDataExtractor on a document, use:

```
cde extract <path>
```

where `path` is the path to an input file in HTML, XML or PDF format. This will write the output to the console. It is also possible to specify an output file using the `-o` option:

```
cde extract <path> -o results.json
```

This will create a file called `results.json` containing the extraction results. Currently, it is only possible to use ChemDataExtractor in its default configuration via the command line interface. For customization, use the Python API.

Reading Documents

ChemDataExtractor processes each document input into a consistent internal format. To see what this looks like, run:

```
cde read <path>
```

where `path` is the path to an input file in HTML, XML or PDF format. This will output a list of document elements.

Tokenization

The first stage in the natural language processing pipeline is tokenization. First, text is split on sentence boundaries. To run the sentence tokenizer on a document, run:

```
cde tokenize sentences <path>
```

This will output each sentence on a new line.

Each sentence is then split into individual word tokens. To do this, run:

```
cde tokenize words <path>
```

This returns an output with further spaces inserted between each token in each sentence line.

Part-of-Speech Tagging

ChemDataExtractor contains a part-of-speech (POS) tagger that has been trained specifically for chemistry text:

```
cde pos tag <path>
```

The output consists of tokens followed by a forward slash and then their POS tag.

1.4.10 Regular Expressions

Regular expressions are an important tool in the Natural Language Processing toolbox. They are special strings that can be used to match sub-strings for the purpose of searching, splitting or grouping text. Regular expressions appear frequently in ChemDataExtractor, most commonly in the chemical property parsers that will be outlined in the next section. Below, we provide a number of useful links for information on Regular Expressions.

If you are unfamiliar with Regular Expressions, I recommend going through the TutorialsPoint Python Regular Expressions tutorial.

Python contains a useful regular expressions library `re` that also contains extensive documentation (https://docs.python.org/3/library/re.html).

Formatting Regular Expressions can be problematic, especially for highly nested groups. Perhaps the most useful tool for dealing with Regular Expressions is Debuggex which provides a beautiful graphical interface for debugging regular expressions.

1.4.11 Creating a New Property Parser

Depending on your specific use case, you will almost definitely need to add new property parsers to ChemDataExtractor in order to retrieve new properties from scientific text/tables. Here we take you through a simple example of how to create a new parser.

First, we need all the relevant imports:

```
from chemdataextractor import Document
from chemdataextractor.model import BaseModel, Compound
from chemdataextractor.model.units import TemperatureModel
from chemdataextractor.doc import Paragraph, Heading
```

Let's create a simple example document with a single heading followed by a single paragraph that contains a boiling point:

```
d = Document(
    Heading(u'Synthesis of 2,4,6-trinitrotoluene (3a)'),
    Paragraph(u'The procedure was followed to yield a pale yellow solid (b.p. 240 °C)
↪')
)
```

By default, ChemDataExtractor wont extract the `boiling_point` property. This can be shown by examining the output records:

```
>>> d.records.serialize()
[{'Compound': {'names': ['2,4,6-trinitrotoluene'], 'labels': ['3a'], 'roles': [
↪'product']}}]
```

So we want to create a *boiling_point* property parser.

Step 1: Defining a new property model

In `chemdataextractor.model.py` you will see all the current property models defined in ChemDataExtractor. Each property inherits from `BaseModel` and can contain fields that can have different types (`StringType`: a string, `ModelType`: Another property model, `ListType`: A list of another type e.g. `ListType(StringType())` is a list of strings).

So in `model.py` we need to create a `BoilingPoint` class and give it some useful fields. As a boiling point is a temperature, we can subclass the `TemperatureModel` class which automatically gives value and unit fields. Now all we need to add is a compound.

```
class BoilingPoint(TemperatureModel):
    """ A boiling point property"""
    compound = ModelType(Compound)
```

Such models automatically have *QuantityModelTemplateParser*, *MultiQuantityModelTemplateParser* set as the sentence parsers and *AutoTableParser* as the table parser. These parsers use the provided information to extract the model defined by the user. In the above case, the user hasn't yet provided any indication of what the property is called, so this will pick up all mentions of temperatures found in the document will be extracted. To make sure that we only find boiling points, we can alter the model as follows:

```
class BoilingPoint(TemperatureModel):
    """ A boiling point property"""
    specifier = StringType(parse_expression=(I('Boiling') + I('Point')).add_
↪action(join), required=True)
    compound = ModelType(Compound)
```

We now have a specifier, which specifies a phrase that must be found in a sentence for the model to be extracted. The parse expression for the specifier is written in the `parse_expression` field, in this case showing that we need to find the word boiling followed by the word point, and the case does not matter. More detail on these parse elements is provided *below*.

Note: If the parse expression is more than one word long, please add the action `join()` to the parse expression so that the whole specifier is picked up by the automatically generated parsers correctly.

Also note the `required` parameter being set to be `True`. The required parameter defines whether a field is required for a model instance to be valid. For example, in the above case, any records without a specifier will be discarded by CDE.

Another parameter which one could set is the `contextual`, which is `False` by default. This parameter defines whether information from other elements of the document will be merged into this field. For example, if we wanted to capture the altitude at which the melting point was captured, we could set up the following:

```
class Altitude(LengthModel):
    specifier = StringType(parse_expression=I('Altitude'), required=True)
    pass

class BoilingPoint(TemperatureModel):
    """ A boiling point property"""
    specifier = StringType(parse_expression=(I('Boiling') + I('Point')).add_
↪action(join), required=True)
    compound = ModelType(Compound)
    pressure = ModelType(Pressure, contextual=True)
```

By doing this, the altitude, which may be found in a different sentence or even a different paragraph, can be added a boiling point record automatically using CDE's interdependency resolution facilities.

If the nested property (e.g. the altitude the above example) is associated with a compound as well, it may be worth adding an associated compound to altitude and making the compound field a binding one:

```
class Altitude(LengthModel):
    specifier = StringType(parse_expression=I('Altitude'), required=True)
    compound = ModelType(Compound)

class BoilingPoint(TemperatureModel):
    """ A boiling point property"""
    specifier = StringType(parse_expression=(I('Boiling') + I('Point')).add_
↪action(join), required=True)
    compound = ModelType(Compound, binding=True)
    pressure = ModelType(Pressure, contextual=True)
```

The `binding` parameter is set to `False` by default, but by setting it to `True`, we can make sure that any fields with the same name in nested fields are consistent. For example, in the above case, it would ensure that the altitude is associated with the same compound as the boiling point.

These three properties, `contextual`, `required`, and `binding`, ensure that CDE's interdependency resolution facilities work as well as possible and are especially important with more complicated models such as those shown above.

Step 2: Writing a Parser

Whilst ChemDataExtractor provides certain automatically generated parsers for properties (for more information on these automatically generated parsers, see *Examples*), one can also write their own parser for higher precision.

Now we need to create the logic that actually extracts boiling points from the text. ChemDataExtractor uses nested rules (*grammars*) to extract chemical properties. These parsers are defined in the `chemdataextractor.parse` package. For example, have a look at the melting point parser in `chemdataextractor.parse.mp_new.py`. This contains a number of statements that are used to define the melting point relationship.

It seems very complicated at first, but let's break the first statement down into its constituent parts:

```
prefix = Optional(I('a')).hide() + (Optional(lbrct) + W('Tm') + Optional(rbrct)| R('^
↪m\.?pt?\.?$', re.I) | I('melting') + Optional((I('point') | I('temperature')| I(
↪'range'))) | R('^m\.?$', re.I) + R('^pt?\.?$', re.I)).hide() + Optional(lbrct + W(
↪'Tm') + rbrct) + Optional(W('=') | I('of') | I('was') | I('is') | I('at')).hide() +␣
↪Optional(I('in') + I('the') + I('range') + Optional(I('of')) | I('about')).hide()
```

Here, we have created a variable prefix, that contains the logic for identifying the melting point relationship specifier (e.g. the text that makes it clear we are talking about a melting point in the text, such "a melting temperature, Tm, "). The grammar contains several elements, with nested logic. Each token must be assigned an element type, these can be:

- `I`: A case insensitive word
- `W`: A case sensitive word
- `R`: A regular expression rule
- `T`: A Part-of-Speech tag

Tokens can be joined using the + symbol, and or logic can be formed using the | symbol.

There are also a number of `ParseElementEnhance` classes that can be used, found in the `chemdataextractor.parse.elements.py` file:

- `Optional`: Matches the contained tokens if they appear, but are not required to form a match
- `ZeroOrMore`: Matches any number of the contained tokens
- `Any`: Matches any token e.g. `ZeroOrMore(Any())` will match the whole of the text
- `OneOrMore`: Similar to zero or more, but at least one token is required.
- `Not`: Looks ahead to disallow a match

Finally, we note that we can hide elements by adding the `.hide()` method. This means that when the parser creates the relationship tree, the hidden tokens are not used.

Continuing to look at the melting point parser, we see the following line:

```
units = (W('°') + Optional(R('^[CFK]\.?$')) | W('K\.?'))('units').add_action(merge)
```

This will clearly match any temperature unit, and as such we tag the rule as 'units'. On top of the tags, we can do some post-processing actions to clean up the output. Here, we add the action `merge`, which joins all tokens without whitespace (° C becomes °C). Other actions include:

- `join`: Join tokens into a single string with spaces between.
- `flatten`: Replace all child results with their text contents.

So now we are able to create our own property parsing rules. Create a file `bp.py` in the parse package. Some very simple logic for extracting boiling points might be:

```
from chemdataextractor.parse import R, I, W, Optional, merge
from chemdataextractor.parse.base import BaseParser
from chemdataextractor.utils import first
```

(continues on next page)

(continued from previous page)

```
prefix = (R(u'^b\.?p\.?$', re.I) | I(u'boiling') + I(u'point')).hide()
units = (W(u'°') + Optional(R(u'^[CFK]\.?$')))(u'raw_units').add_action(merge)
value = R(u'^\d+(\.\d+)?$')(u'raw_value')
bp = (prefix + value + units)(u'bp')
```

The most important thing to note is that the final phrase (called `bp`) is now a nested tree, with tags labelling the elements. If we were to reproduce the XML it would look like:

```
<bp>
    <value>R(u'^\d+(\.\d+)?$')</value>
    <units>W(u'°') + Optional(R(u'^[CFK]\.?$'))</units>
</bp>
```

Now we have to create the logic for parsing this structure. In the same file, we create the parser class, that inherits from `BaseParser`:

```
class BpParser(BaseSentenceParser):
    root = bp

    def interpret(self, result, start, end):
        try:
            raw_value = first(result.xpath('./value/text()'))
            raw_units = first(result.xpath('./units/text()'))
            boiling_point = self.model(raw_value=raw_value,
                        raw_units=raw_units,
                        value=self.extract_value(raw_value),
                        error=self.extract_error(raw_value),
                        units=self.extract_units(raw_units, strict=True))
            yield boiling_point
        except TypeError as e:
            log.debug(e)
```

All parser classes must define:

- A root variable: i.e. the phrase that forms the head of the tree
- An *interpret* function: That defines the parsing logic

The *interpret* function then creates a new compound (with the model we defined in `model.py`) and adds a boiling point property. Here, the result parameter is the result of the parsing process. If a tree with root bp is found, we access the value and unit elements using XPath expressions.

Note: CDE also provides an automatic interpret function if you subclass from *`BaseAutoParser`*. This interpret function relies upon all the names of the tags in the parse expressions being the same as the names of the fields in the model.

Finally, we need to tell ChemDataExtractor to parse the `BoilingPoint` model with the newly written parser. This can be done by setting the parsers associated with the `BoilingPoint` model:

```
BoilingPoint.parsers = [BpParser()]
```

alternatively, we could have this parser in addition to the default parsers:

```
BoilingPoint.parsers.append(BpParser())
```

Step 3: Testing the Parser

Now we can simply re-run the document through ChemDataExtractor:

```
>>> d = Document(
>>>     Heading(u'Synthesis of 2,4,6-trinitrotoluene (3a)'),
>>>     Paragraph(u'The procedure was followed to yield a pale yellow solid (b.p. 240
↪°C)')
>>>     )

>>> d.records.serialize()
[{'BoilingPoint': {'raw_value': '240',
                   'raw_units': '°C',
                    'compound': {'Compound': {'names': ['2,4,6-trinitrotoluene'],
↪'labels': ['3a'], 'roles': ['product']}}}}]
```

Of course, real world examples are much more complex than this, and a large amount of trial and error is needed to create good parsers. It should also be noted that in this example, the chemical label ('3a') is found using interdependency resolution between the heading and associated paragraph. In some cases you will need to put the chemical labels and names directly into the parser. Rules for chemical entity recognition can be found in `chemdataextractor.parse.cem.py`.

Table Parsers

ChemDataExtractor parses tables in a similar way. In `chemdataextractor.parse.table.py` you will find the logic for finding chemical relationships from tables. These parsers can be written very similarly to a sentence parser, but require the parser to be subclassed from *`BaseTableParser`* instead of *`BaseSentenceParser`*.

However, due to the relatively uniform nature of tables and TableDataExtractor's powerful table normalisation facilities, the automatically generated parser for tables tend to perform very well, with precisions of over 90% for tables often being achievable by choosing the right parse expressions and setting the `required`, `contextual` and `binding` properties appropriately.

1.4.12 ChemDataExtractor REST API

A web service for programmatically uploading documents to be processed using ChemDataExtractor on our servers.

All endpoints are at constructed by appending to http://chemdataextractor.org/api

1.4.13 "Where do I find...?"

The most common questions about ChemDataExtractor usually involve trying to find functionality or asking where best to put new functionality. Below is a list of the general roles each of the packages perform:

- `biblio`: Misc tools for parsing bibliographic information such as bibtex files, author names etc.
- `cli`: Command line interfact tools
- `doc`: Logic for reading/creating documents. That is, splitting documents down into its various elements.
- `nlp`: Tools for performing the NLP stages, such as POS tagging, Word clustering, CNER, Abbreviation detection
- `parse`: Chemical property parsers
- `Reader`: Document readers

- `scrape`: Scrapers for the various data sources
- `text`: Useful tools for processing text

If you have new functionality that doesn't fit into one of these categories you may want to create a new sub-package. Alternatively, if your new functionality is very specific to your own use case, it may be better to have it external to ChemDataExtractor.

1.5 Contributing

Contributions of any kind are greatly appreciated!

Feedback & Discussion

The Issue Tracker is the best place to post any feature ideas, requests and bug reports. This way, everyone has the opportunity to keep informed of changes and join the discussion on future plans.

Contributing Code

If you are able to contribute changes yourself, just fork the source code on GitHub (https://github.com/CambridgeMolecularEngineering/chemdataextractor), make changes and file a pull request. All contributions are welcome, no matter how big or small.

The following are especially welcome:

- New document readers for patent formats and the website HTML of scientific publishers.
- Improvements to NLP components - tokenization, tagging and entity recognition.
- Parsers for extracting new compound properties.
- New or improved documentation of existing features.

Quick Guide to Contributing

1. Fork the ChemDataExtractor repository on GitHub, then clone your fork to your local machine:

   ```
   git clone https://github.com/<your-username>/ChemDataExtractor.git
   ```

2. Install the development requirements:

   ```
   cd ChemDataExtractor
   pip install -r requirements/development.txt
   ```

3. Create a new branch for your changes:

   ```
   git checkout -b <name-for-branch>
   ```

4. Make your changes or additions. Ideally add some tests and ensure they pass by running:

   ```
   pytest
   ```

   The output should show all tests passing.

5. Commit your changes and push to your fork on GitHub:

```
git add .
git commit -m "<description-of-changes>"
git push origin <name-for-branch>
```

4. Submit a pull request. Then we can discuss your changes and merge them into the main ChemDataExtractor repository.

Tips

- Follow the PEP8 style guide.
- Include docstrings as described in PEP257.
- Try and include tests that cover your changes.
- Try to write good commit messages.
- Read the GitHub help page on Using pull requests.

1.5.1 Documenting Code

Everyone is encouraged to contribute to documentation in the form of tutorial sections, examples and in any other way that will improve it.

When you are adding a section of documentation to the `.rst` files add you name to it, with:

```
.. sectionauthor:: My Name <my.name@email.com>
```

If you are adding documentation in the source code (docstrings and boilerplates), the correct form is:

```
.. codeauthor:: My Name <my.name@email.com>
```

All new code should be documented in the docstrings of the appropriate modules, functions an classes, using `.rst` format. In this way, documentation will be automatically created using Sphinx (see *API Documentation*).

Note: You can add docstrings for a one-line function/object using `#:`, preceding the definition. This is particularly useful for documenting regular expressions in `chemdataextractor.parse.cem.py`. For example:

```
#: Amino acid abbreviations. His removed, too ambiguous
amino_acid = R('^
↪((Ala|Arg|Asn|Asp|Cys|Glu|Gln|Gly|Ile|Leu|Lys|Met|Phe|Pro|Ser|Thr|Trp|Tyr|Val)-?)+$
↪')
```

will create a correct documentation entry.

If you are adding new modules (.py files) to the codebase, make sure they are included in the documentation (check some of the example files in `docs/source_code_docs/`. Most importantly, add an `.. autosummary::` to `code_documentation.rst` and a file describing all the new content of the module (new classes and functions).

Note: Private methods are not included by default in the documentation! Functions that are decorated and are not members of a class have to be included into the documentation manually with:

```
.. autofunction:: decorated_function(parameters)
```

Additional *tutorial-like* content can be added by hand in the appropriate `.rst` files. When you are writing headers in `.rst`, use the python convention:

- `#` with overline, for parts
- `*` with overline, for chapters
- `=`, for sections
- `-`, for subsections
- `^`, for subsubsections
- `"`, for paragraphs

For highlighted paragraph heading that you don't wish to include into the toctree use `.. rubric:: My heading`. Check out the source documentation `.rst` files for inspiration on how to use `.rst` for code-blocks and other features. *It's made to be very simple!*

Parameters for compiling the html documentation with spinx are:

- command: `html`
- input: `/chemdataextractor-development/docs`
- output: `/chemdataextractor-development/docs/_build/html`
- options: optionally, use `-a` and `-E` to build the documentation from scratch.
- working directory: `/chemdataextractor-development/docs`

So, in the bash shell, from within the working directory you would use:

```
$ sphinx-build -b html docs docs/_build/html
```

However, it is encouraged to set up a Sphinx Run configuration in the IDE you are using for development. It is very easy to do in Pycharm, where you can run sphinx within the same Python virtual environment you are using for the development of ChemDataExtractor.

The `conf.py` file is used to set-up internal sphinx parameters. Change it with caution!

CHAPTER 2

Examples

2.1 Creating new units and dimensions

2.1.1 Overview

Units and dimensions are a new addition to ChemDataExtractor 2.0, and form a fundamental component in making sure that a large number of new features, such as automatic parsing, works. Many units are included in ChemDataExtractor, making it easy to create models out of the box, but there may be cases when you need to create your own units, and this document should get you started with that.

2.1.2 Dimensions

If you're implementing a completely new type of unit, then the first step you need to take is to write a dimension for it. An example of an implementation of dimension is as follows:

```
class Temperature(Dimension):
    pass
```

As you can see, it's incredibly simple to define a dimension from scratch; all that really matters is the name of the dimension, so you just need to define the name for the dimension and the rest can be empty. If you want to implement *composite dimensions*, that is, dimensions which are composed of other, more basic dimensions, such as speed, you just need one more line of code:

```
class Speed(Dimension):
    constituent_dimensions = Length() / Time()
```

2.1.3 Units

Defining a unit for a certain dimension is also straightforward, but each unit needs to implement an `__init__` function, although it's just boilerplate code:

```
def __init__(self, magnitude=0.0, powers=None):
    super(TemperatureUnit, self).__init__(Temperature(), magnitude, powers)
```

Where the first argument passed to the superclass should be the dimensions that you want the unit to have. Writing this for each unit would be wasteful, so for each commonly used type of unit, we have defined a subclass of *Unit*, such as `TemperatureUnit`, which you can subclass from to get these initializers for free. We would encourage you to do the same.

Once that's done, each unit needs to implement functions to convert values and errors to the standard value. The four functions that need to be implemented are *convert_value_to_standard()*, *convert_value_from_standard()*, *convert_error_to_standard()*, and *convert_error_from_standard()*. It is crucial that you provide documentation about what the standard unit is for each type of unit so that when people build other units of the same type, they know what each function should do. An example of a unit is the Fahrenheit class:

```
class Fahrenheit(TemperatureUnit):
    """
    Class for Fahrenheit.
    """

    def convert_value_to_standard(self, value):
        return (value + 459.67) * (5. / 9.)

    def convert_value_from_standard(self, value):
        return value * (9. / 5.) - 459.67

    def convert_error_to_standard(self, error):
        return error * (5. / 9.)

    def convert_error_from_standard(self, error):
        return error * (9. / 5.)
```

Defining a standard unit

In addition to documenting what the standard unit is, you should also enforce it in code by setting the dimension's standard units. This can simply be done as follows, in the example of temperature:

```
Temperature.standard_units = Kelvin()
```

After which all instances of temperature will hold a reference to the correct standard units. This is used in *convert_to_standard()* to make it easy to convert any models to the standard values. If you define a composite dimension and this property not set, the standard units will be automatically inferred from the constituent units' standard units, e.g. a speed dimension will automatically have a standard unit of m/s.

2.1.4 Adding facilities for parsing

Whilst all magnitudes (e.g. kilo-, centi-, mega-, etc.) are handled by ChemDataExtractor, you need to write down parse expressions to make sure that units are extracted correctly and picked up correctly by autoparsers. To do this, you need to set the `units_dict` property of your dimension. An example can be seen in the case of temperatures:

```
units_dict = {R('°?(((K|k)elvin(s)?)|K)\.?', group=0): Kelvin,
              R('(°C|((C|c)elsius))\.?', group=0): Celsius,
              R('°?((F|f)ahrenheit|F)\.?', group=0): Fahrenheit,
```

(continues on next page)

(continued from previous page)

```
          R('°|C', group=0): None}
Temperature.units_dict = units_dict
```

Note: This property needs to be set after the declaration of the dimension class as `units_dict` references the units which in turn reference the dimensions.

Note: The units_dict has been extensively tested using regex elements, and while in theory it may work with other parse elements, it is strongly recommended that you use a regex element. If a regex element is specified, it should

- Not have a $ symbol at the end: the units can be passed in with numbers or other symbols after it, and these are also used in the autoparser to find candidate tokens which may contain units, and a $ symbol at the end would stop this from working.
- Have the group attribute set to 0. Unless this is set, the default behaviour of the regex element is to return the whole token in which the match was found. This is unhelpful behaviour for our logic for extracting units, as we want to extract only the exact characters that matched the unit.

The final element in the above *units_dict* has no unit set to it. This is a special case which is used by autoparsers but not during units extraction. This is to handle the fact that °C is always split into two tokens, so we need to be able to capture these separately, but we do not want that to affect units extraction later down the line.

In the case you have a composite dimension, such as energy, you should **update** *units_dict* instead of setting it, so that ChemDataExtractor can correctly extract the dimension even if it is composed of its constituent SI units.

```
units_dict = {R('(J|j)(oule(s)?)?', group=0): Joule,
              R('(E|e)(lectron)( )?(V|v)(olts)?', group=0): ElectronVolt}
Energy.units_dict.update(units_dict)
```

Check out the Jupyter notebook examples in the source code!

CHAPTER 3

API Documentation

Note: Private methods are not included in the documentation!

3.1 chemdataextractor

3.1.1 .config

Config file reader/writer.

chemdataextractor.config.**construct_yaml_str**(*self*, *node*)
Override the default string handling function to always return unicode objects.

class chemdataextractor.config.**Config**(*path=None*)
Bases: collections.abc.MutableMapping

Read and write to config file.

A config object is essentially a string key-value store that can be treated like a dictionary:

```
c = Config()
c['foo'] = 'bar'
print c['foo']
```

The file location may be specified:

```
c = Config('~/matt/anotherconfig.yml')
c['where'] = 'in a different file'
```

If no location is specified, the environment variable CHEMDATAEXTRACTOR_CONFIG is checked and used if available. Otherwise, a standard config location is used, which varies depending on the operating system. You can check the location using the `path` property. For more information see https://github.com/ActiveState/appdirs

It is possible to edit the file by hand with a text editor. It is in YAML format.

Warning: multiple instances of Config() pointing to the same file will not see each others' changes, and will overwrite the entire file when any key is changed.

__init__(*path=None*)

Parameters **path** (*string*) – (Optional) Path to config file location.

path
The path to the config file.

clear()
Clear all values from config.

chemdataextractor.config.**config = <Config: /home/docs/.config/ChemDataExtractor/chemdatae**
Global config instance.

3.1.2 .data

Tools for loading and caching data files.

class chemdataextractor.data.**Package**(*path*)
Bases: object

Data package.

__init__(*path*)
Initialize self. See help(type(self)) for accurate signature.

remote_path

local_path

remote_exists()

local_exists()

download(*force=False*)

chemdataextractor.data.**PACKAGES = [<Package: models/cem_crf-1.0.pickle>, <Package: model**
Current active data packages

chemdataextractor.data.**get_data_dir**()
Return path to the data directory.

chemdataextractor.data.**find_data**(*path*, *warn=True*)
Return the absolute path to a data file within the data directory.

chemdataextractor.data.**load_model**(*path*)
Load a model from a pickle file in the data directory. Cached so model is only loaded once.

3.1.3 .errors

Error classes for ChemDataExtractor.

exception chemdataextractor.errors.**ChemDataExtractorError**
Bases: Exception

Base ChemDataExtractor exception.

exception chemdataextractor.errors.**ReaderError**
Bases: *chemdataextractor.errors.ChemDataExtractorError*

Raised when a reader is unable to read a document.

exception chemdataextractor.errors.**ModelNotFoundError**
Bases: *chemdataextractor.errors.ChemDataExtractorError*

Raised when a model file could not be found.

3.1.4 .utils

Miscellaneous utility functions.

chemdataextractor.utils.**memoized_property**(*fget*)
Decorator to create memoized properties.

chemdataextractor.utils.**memoize**(*obj*)
Decorator to create memoized functions, methods or classes.

chemdataextractor.utils.**python_2_unicode_compatible**(*klass*)
Fix __str__, __unicode__ and __repr__ methods under Python 2.

class chemdataextractor.utils.**Singleton**
Bases: type

Singleton metaclass.

chemdataextractor.utils.**flatten**(*x*)
Return a single flat list containing elements from nested lists.

chemdataextractor.utils.**first**(*el*)

chemdataextractor.utils.**ensure_dir**(*path*)
Ensure a directory exists.

3.2 .biblio

Misc tools for parsing bibliographic information such as bibtex files, author names etc.

Tools for dealing with bibliographic information.

3.2.1 .biblio.bibtex

BibTeX parser.

class chemdataextractor.biblio.bibtex.**BibtexParser**(*data*, ***kwargs*)
Bases: object

A class for parsing a BibTeX string into JSON or a python data structure.

Example usage:

```
with open(example.bib, 'r') as f:
    bib = BibtexParser(f.read())
    bib.parse()
    print bib.records_list
    print bib.json
```

__init__(*data*, ***kwargs*)
Initialize BibtexParser with data.

Optional metadata passed as keyword arguments will be included in the JSON output. e.g. collection, label, description, id, owner, created, modified, source

Example usage:

```
bib = BibtexParser(data, created=unicode(datetime.utcnow()), owner='mcs07')
```

parse()
Parse self.data and store the parsed BibTeX to self.records.

classmethod parse_names(*names*)
Parse a string of names separated by "and" like in a BibTeX authors field.

size
Return the number of records parsed.

records_list
Return the records as a list of dictionaries.

metadata
Return metadata for the parsed collection of records.

json
Return a list of records as a JSON string. Follows the BibJSON convention.

chemdataextractor.biblio.bibtex.**parse_bibtex**(*data*)

3.2.2 .biblio.person

Tools for parsing people's names from strings into various name components.

class chemdataextractor.biblio.person.**PersonName**(*fullname=None*, *from_bibtex=False*)
Bases: dict

Class for parsing a person's name into its constituent parts.

Parses a name string into title, firstname, middlename, nickname, prefix, lastname, suffix.

Example usage:

```
p = PersonName('von Beethoven, Ludwig')
```

PersonName acts like a dict:

```
print p
print p['firstname']
print json.dumps(p)
```

Name components can also be access as attributes:

```
print p.lastname
```

Instances can be reused by setting the name property:

```
p.name = 'Henry Ford Jr. III'
print p
```

Two PersonName objects are equal if every name component matches exactly. For fuzzy matching, use the *could_be* method. This returns True for names that are not explicitly inconsistent.

This class was written with the intention of parsing BibTeX author names, so name components enclosed within curly brackets will not be split.

__init__ (*fullname=None*, *from_bibtex=False*)
Initialize with a name string.

Parameters

- **fullname** (*str*) – The person's name.
- **from_bibtex** (*bool*) – (Optional) Whether the fullname parameter is in BibTeX format. Default False.

could_be (*other*)
Return True if the other PersonName is not explicitly inconsistent.

fullname

3.2.3 .biblio.xmp

Parse metadata stored as XMP (Extensible Metadata Platform).

This is commonly embedded within PDF documents, and can be extracted using the PDFMiner framework.

More information is available on the Adobe website:

http://www.adobe.com/products/xmp/index.html

class chemdataextractor.biblio.xmp.**XmpParser**(*ns_map={'http://crossref.org/crossmark/1.0/': 'crossmark', 'http://ns.adobe.com/pdf/1.3/': 'pdf', 'http://ns.adobe.com/pdfx/1.3/': 'pdfx', 'http://ns.adobe.com/xap/1.0/': 'xap', 'http://ns.adobe.com/xap/1.0/mm/': 'xapmm', 'http://ns.adobe.com/xap/1.0/rights/': 'rights', 'http://prismstandard.org/namespaces/basic/2.0/': 'prism', 'http://purl.org/dc/elements/1.1/': 'dc', 'http://www.w3.org/1999/02/22-rdf-syntax-ns#': 'rdf', 'http://www.w3.org/XML/1998/namespace': 'xml'}*)

Bases: object

A parser that converts an XMP metadata string into a python dictionary.

Usage:

```
parser = XmpParser()
metadata = parser.parse(xmpstring)
```

Common namespaces are abbreviated in the output using the definitions in xmp.NS_MAP. If an abbreviation for a namespace is not defined in NS_MAP, the full URL is used as the key in the output dictionary. It is possible to override NS_MAP when initializing the parser:

```
parser = XmpParser(ns_map={'http://www.w3.org/XML/1998/namespace': 'xml'})
metadata = parser.parse(xmpstring)
```

__init__(*ns_map={'http://crossref.org/crossmark/1.0/': 'crossmark', 'http://ns.adobe.com/pdf/1.3/': 'pdf', 'http://ns.adobe.com/pdfx/1.3/': 'pdfx', 'http://ns.adobe.com/xap/1.0/': 'xap', 'http://ns.adobe.com/xap/1.0/mm/': 'xapmm', 'http://ns.adobe.com/xap/1.0/rights/': 'rights', 'http://prismstandard.org/namespaces/basic/2.0/': 'prism', 'http://purl.org/dc/elements/1.1/': 'dc', 'http://www.w3.org/1999/02/22-rdf-syntax-ns#': 'rdf', 'http://www.w3.org/XML/1998/namespace': 'xml'}*)
Initialize self. See help(type(self)) for accurate signature.

parse(*xmp*)
Run parser and return a dictionary of all the parsed metadata.

chemdataextractor.biblio.xmp.**parse_xmp**(*xmp*)
Shorthand function for parsing an XMP string into a python dictionary.

3.3 .cli

Command line interface tools

ChemDataExtractor command line interface.

Once installed, ChemDataExtractor provides a command-line tool that can be used by typing 'cde' in a terminal.

chemdataextractor.cli.**cli**(*ctx*, *verbose*)
ChemDataExtractor command line interface.

chemdataextractor.cli.**extract**(*ctx*, *input*, *output*)
Run ChemDataExtractor on a document.

chemdataextractor.cli.**read**(*ctx*, *input*, *output*)
Output processed document elements.

3.3.1 .cli.cem

Chemical entity mention (CEM) commands.

chemdataextractor.cli.cem.**cem**(*ctx*)
Chemical NER commands.

chemdataextractor.cli.cem.**train_crf**(*ctx*, *input*, *output*, *clusters*)
Train CRF CEM recognizer.

3.3.2 .cli.chemdner

Command line tools for dealing with CHEMDNER corpus.

chemdataextractor.cli.chemdner.**chemdner_cli**(*ctx*)
CHEMDNER commands.

chemdataextractor.cli.chemdner.**prepare_gold**(*ctx*, *annotations*, *gout*)
Prepare bc-evaluate gold file from annotations supplied by CHEMDNER.

chemdataextractor.cli.chemdner.**prepare_tokens**(*ctx*, *input*, *annotations*, *tout*, *lout*)
Prepare tokenized and tagged corpus file from those supplied by CHEMDNER.

chemdataextractor.cli.chemdner.**tag**(*ctx*, *corpus*, *output*)
Tag chemical entities and write CHEMDNER annotations predictions file.

3.3.3 .cli.cluster

Word clusters command-line interface.

chemdataextractor.cli.cluster.**cluster_cli**(*ctx*)
Word clusters commands.

chemdataextractor.cli.cluster.**load**(*ctx*, *input*, *output*)
Read clusters from file and save to model file.

3.3.4 .cli.config

Commands for managing ChemDataExtractor configuration.

chemdataextractor.cli.config.**config_cli**(*ctx*)
Manage configuration.

chemdataextractor.cli.config.**list**(*ctx*)
List all config values.

chemdataextractor.cli.config.**get**(*ctx*)
Get the config value for a key.

chemdataextractor.cli.config.**set**(*ctx*, *key*, *value*)
Set the config value for a key.

chemdataextractor.cli.config.**remove**(*ctx*, *key*)
Remove the config value for a key.

chemdataextractor.cli.config.**clear**(*ctx*)
Clear all config values.

3.3.5 .cli.data

Data and model management interface.

chemdataextractor.cli.data.**data_cli**(*ctx*)
Data and model management commands.

chemdataextractor.cli.data.**where**(*ctx*)
Print path to data directory.

chemdataextractor.cli.data.**list**(*ctx*)
List active data packages.

chemdataextractor.cli.data.**download**(*ctx*)
Download data.

chemdataextractor.cli.data.**clean**(*ctx*)
Prune data that is no longer required.

3.3.6 .cli.dict

Commands for building a dictionary-based chemical named entity recognizer.

chemdataextractor.cli.dict.**dict_cli**(*ctx*)
Chemical dictionary commands.

chemdataextractor.cli.dict.**prepare_jochem**(*ctx*, *jochem*, *output*, *csoutput*)
Process and filter jochem file to produce list of names for dictionary.

chemdataextractor.cli.dict.**prepare_include**(*ctx*, *include*, *output*)
Process and filter include file to produce list of names for dictionary.

chemdataextractor.cli.dict.**build**(*ctx*, *inputs*, *output*, *cs*)
Build chemical name dictionary.

chemdataextractor.cli.dict.**tag**(*ctx*, *model*, *cs*, *corpus*, *output*)
Tag chemical entities and write CHEMDNER annotations predictions file.

3.3.7 .cli.evaluate

Commands for running evaluations.

chemdataextractor.cli.evaluate.**evaluate**(*ctx*)
Evaluation commands.

chemdataextractor.cli.evaluate.**run**(*input*)

chemdataextractor.cli.evaluate.**compare**()

chemdataextractor.cli.evaluate.**eval_document**(*gold*, *out*, *transform=None*)

chemdataextractor.cli.evaluate.**get_names**(*cs*)
Return list of every name.

chemdataextractor.cli.evaluate.**get_labels**(*cs*)
Return list of every label.

chemdataextractor.cli.evaluate.**get_ids**(*cs*)
Return chemical identifier records.

chemdataextractor.cli.evaluate.**get_spectra_type**(*cs*)

chemdataextractor.cli.evaluate.**get_spectra_subject**(*cs*)

chemdataextractor.cli.evaluate.**get_spectra_peaks**(*cs*)

chemdataextractor.cli.evaluate.**get_spectra_solvent**(*cs*)

chemdataextractor.cli.evaluate.**get_spectra_core**(*cs*)

chemdataextractor.cli.evaluate.**get_spectra_temp**(*cs*)

chemdataextractor.cli.evaluate.**get_spectra_apparatus**(*cs*)

chemdataextractor.cli.evaluate.**get_spectra_full**(*cs*)

chemdataextractor.cli.evaluate.**get_property_value**(*cs*)

chemdataextractor.cli.evaluate.**get_property_units**(*cs*)

chemdataextractor.cli.evaluate.**get_property_subject**(*cs*)

chemdataextractor.cli.evaluate.**get_property_solvent**(*cs*)

chemdataextractor.cli.evaluate.**get_property_temperature**(*cs*)

chemdataextractor.cli.evaluate.**get_property_apparatus**(*cs*)

chemdataextractor.cli.evaluate.**get_property_core**(*cs*)

chemdataextractor.cli.evaluate.**get_property_full**(*cs*)

3.3.8 .cli.pos

Part of speech tagging commands.

chemdataextractor.cli.pos.**pos_cli**(*ctx*)
POS tagger commands.

chemdataextractor.cli.pos.**train_all**(*ctx*, *output*)
Train POS tagger on WSJ, GENIA, and both. With and without cluster features.

chemdataextractor.cli.pos.**evaluate_all**(*ctx*, *model*)
Evaluate POS taggers on WSJ and GENIA.

chemdataextractor.cli.pos.**train**(*ctx*, *output*, *corpus*, *clusters*)
Train POS Tagger.

chemdataextractor.cli.pos.**evaluate**(*ctx*, *model*, *corpus*, *clusters*)
Evaluate performance of POS Tagger.

chemdataextractor.cli.pos.**train_perceptron**(*ctx*, *output*, *corpus*, *clusters*)
Train Averaged Perceptron POS Tagger.

chemdataextractor.cli.pos.**evaluate_perceptron**(*ctx*, *model*, *corpus*)
Evaluate performance of Averaged Perceptron POS Tagger.

chemdataextractor.cli.pos.**tag**(*ctx*, *input*, *output*)
Output POS-tagged tokens.

3.3.9 .cli.tokenize

Tokenizer command line interface.

chemdataextractor.cli.tokenize.**tokenize_cli**(*ctx*)
Tokenizer commands.

chemdataextractor.cli.tokenize.**train_punkt**(*ctx*, *input*, *output*, *abbr*, *colloc*)
Train Punkt sentence splitter using sentences in input.

chemdataextractor.cli.tokenize.**sentences**(*ctx*, *input*, *output*)
Read input document, and output sentences.

chemdataextractor.cli.tokenize.**words**(*ctx*, *input*, *output*)
Read input document, and output words.

3.4 .doc

Logic for reading/creating documents. That is, splitting documents down into its various elements. The API for documents has been slightly changed as of version 1.5.0. Please refer to the *migration guide* and the examples for an overview of the changes.

Document processing.

3.4.1 .doc.document

Document model.

class chemdataextractor.doc.document.**BaseDocument**
Bases: collections.abc.Sequence

Abstract base class for a Document.

elements
Return a list of document elements.

records
Chemical records that have been parsed from this Document.

class chemdataextractor.doc.document.**Document**(**elements*, ***kwargs*)
Bases: *chemdataextractor.doc.document.BaseDocument*

A document to extract data from. Contains a list of document elements.

__init__(**elements*, ***kwargs*)
Initialize a Document manually by passing one or more Document elements (Paragraph, Heading, Table, etc.)

Strings that are passed to this constructor are automatically wrapped into Paragraph elements.

Parameters **elements** (*list[chemdataextractor.doc.element.BaseElement|string]*) – Elements in this Document.

Keyword Arguments

- **config** (Config) – (Optional) Config file for the Document.
- **models** (list[BaseModel]) – (Optional) Models that the Document should extract data for.

add_models(*models*)
Add models to all elements.

Usage:

```
d = Document.from_file(f)
d.set_models([myModelClass1, myModelClass2,..])
```

Arguments:: models – List of model classes

models

classmethod from_file(*f*, *fname=None*, *readers=None*)
Create a Document from a file.

Usage:

```
with open('paper.html', 'rb') as f:
    doc = Document.from_file(f)
```

Note: Always open files in binary mode by using the 'rb' parameter.

Parameters

- **f** (*file or str*) – A file-like object or path to a file.
- **fname** (*str*) – (Optional) The filename. Used to help determine file format.

- **readers** (*list[chemdataextractor.reader.base.BaseReader]*) – (Optional) List of readers to use. If not set, Document will try all default readers, which are *AcsHtmlReader*, *RscHtmlReader*, *NlmXmlReader*, *UsptoXmlReader*, *CsspHtmlReader*, ElsevierXmlReader, *XmlReader*, *HtmlReader*, *PdfReader*, and *PlainTextReader*.

classmethod from_string(*fstring*, *fname=None*, *readers=None*)

Create a Document from a byte string containing the contents of a file.

Usage:

```
contents = open('paper.html', 'rb').read()
doc = Document.from_string(contents)
```

Note: This method expects a byte string, not a unicode string (in contrast to most methods in ChemDataExtractor).

Parameters

- **fstring** (*bytes*) – A byte string containing the contents of a file.
- **fname** (*str*) – (Optional) The filename. Used to help determine file format.
- **readers** (*list[chemdataextractor.reader.base.BaseReader]*) – (Optional) List of readers to use. If not set, Document will try all default readers, which are *AcsHtmlReader*, *RscHtmlReader*, *NlmXmlReader*, *UsptoXmlReader*, *CsspHtmlReader*, ElsevierXmlReader, *XmlReader*, *HtmlReader*, *PdfReader*, and *PlainTextReader*.

elements

A list of all the elements in this document. All elements subclass from *BaseElement*, and represent things such as paragraphs or tables, and can be found in *chemdataextractor.doc.figure*, *chemdataextractor.doc.table*, and *chemdataextractor.doc.text*.

records

All records found in this Document, as a list of *BaseModel*.

get_element_with_id(*id*)

Get element with the specified ID. If one is not found, None is returned.

Parameters **id** – Identifier to search for.

Returns Element with specified ID

Return type *BaseElement* or None

figures

A list of all *Figure* elements in this Document.

tables

A list of all *Table* elements in this Document.

citations

A list of all *Citation* elements in this Document.

footnotes

A list of all *Footnote* elements in this Document.

Note: Elements (e.g. Tables) can contain nested Footnotes which are not taken into account.

titles
A list of all *Title* elements in this Document.

headings
A list of all *Heading* elements in this Document.

paragraphs
A list of all *Paragraph* elements in this Document.

captions
A list of all *Caption* elements in this Document.

captioned_elements
A list of all *CaptionedElement* elements in this Document.

metadata
Return metadata information

abbreviation_definitions
A list of all abbreviation definitions in this Document. Each abbreviation is in the form (str abbreviation, str long form of abbreviation, str ner_tag)

ner_tags
A list of all Named Entity Recognition tags in this Document. If a word was found not to be a named entity, the named entity tag is None, and if it was found to be a named entity, it can have either a tag of 'B-CM' for a beginning of a mention of a chemical or 'I-CM' for the continuation of a mention.

cems
A list of all Chemical Entity Mentions in this document as *Span*

definitions
Return a list of all recognised definitions within this Document

serialize()
Convert Document to Python dictionary. The dictionary will always contain the key 'type', which will be 'document', and the key 'elements', which contains a dictionary representation of each of the elements of the document.

to_json(**args*, ***kwargs*)
Convert Document to JSON string. The content of the JSON will be equivalent to that of *serialize()*. The document itself will be under the key 'elements', and there will also be the key 'type', which will always be 'document'. Any arguments for json.dumps() can be passed into this function.

3.4.2 .doc.element

Document elements.

class chemdataextractor.doc.element.**BaseElement**(*document=None*, *references=None*, *id=None*, *models=None*, ***kwargs*)

Bases: object

Abstract base class for a Document Element.

Variables

- **id** – (Optional) An identifier for this Element.

- **models** (*list[chemdataextractor.models.BaseModel]*) – A list of models that this element will parse

__init__(*document=None*, *references=None*, *id=None*, *models=None*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its *document* attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the *document* attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **document** (Document) – (Optional) The document containing this element.
- **references** (*list[Citation]*) – (Optional) Any references contained in the element.
- **id** (Any) – (Optional) An identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *chemdataextractor.doc.document.Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

document
The *chemdataextractor.doc.document.Document* that this element belongs to.

records
All records found in this Document, as a list of *chemdataextractor.model.base.BaseModel*.

add_models(*models*)
Set all models on this element

models

to_json(**args*, ***kwargs*)
Convert element to JSON string. The content of the JSON will be equivalent to that of `serialize()`.

class chemdataextractor.doc.element.**CaptionedElement**(*caption*, *label=None*, ***kwargs*)

Bases: *chemdataextractor.doc.element.BaseElement*

Document Element with a caption.

Variables ***caption*** (BaseElement) – The caption for this element.

__init__(*caption*, *label=None*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its *document* attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the *document* attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **caption** (BaseElement) – The caption for the element.
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

document
The *Document* that this element belongs to.

records
All records found in the object, as a list of *BaseModel*.

abbreviation_definitions
A list of all abbreviation definitions in this Document. Each abbreviation is in the form (str abbreviation, str long form of abbreviation, str ner_tag)

ner_tags
A list of all Named Entity Recognition tags in the caption for this element. If a word was found not to be a named entity, the named entity tag is None, and if it was found to be a named entity, it can have either a tag of 'B-CM' for a beginning of a mention of a chemical or 'I-CM' for the continuation of a mention.

cems
A list of all Chemical Entity Mentions in this document as *Span*

definitions
Return a list of all specifier definitions in the caption

Returns list– The specifier definitions

chemical_definitions

models

serialize()
Convert self to a dictionary. The key 'type' will contain the name of the class being serialized, and the key 'caption' will contain a serialized representation of caption, which is a *BaseElement*

3.4.3 .doc.figure

Figure document elements. :codeauthor:: Callum Court (cc889@cam.ac.uk)

class chemdataextractor.doc.figure.**Figure**(*caption*, *label=None*, *links=None*, *models=None*, ***kwargs*)
Bases: *chemdataextractor.doc.element.CaptionedElement*

__init__(*caption*, *label=None*, *links=None*, *models=None*, ***kwargs*)
Create a new Figure element, to interface with FDE

records
Return FigureData records

Returns [type] – [description]

3.4.4 .doc.meta

MetaData Document elements

class chemdataextractor.doc.meta.**MetaData**(*data*)

Bases: *chemdataextractor.doc.element.BaseElement*

__init__(*data*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its `document` attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the `document` attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **document** (Document) – (Optional) The document containing this element.
- **references** (*list[Citation]*) – (Optional) Any references contained in the element.
- **id** (Any) – (Optional) An identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *chemdataextractor.doc.document.Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

records
All records found in this Document, as a list of *chemdataextractor.model.base.BaseModel*.

serialize()

title
The article title

authors
The article Authors type:: list()

publisher
The source publisher

journal
The source journal

volume
The source volume

issue
The source issue

firstpage
The source first page title

lastpage
The source last page

doi
The source DOI

pdf_url
The source url to the PDF version

html_url
The source url to the HTML version

date
The source publish date

data
Returns all data as a dict()

abbreviation_definitions

definitions

chemical_definitions

cems

is_unidentified

3.4.5 .doc.table

Table document elements

class chemdataextractor.doc.table.**Table**(*caption*, *label=None*, *table_data=[]*, *models=None*, ***kwargs*)

Bases: *chemdataextractor.doc.element.CaptionedElement*

Main Table object. Relies on TableDataExtractor.

__init__(*caption*, *label=None*, *table_data=[]*, *models=None*, ***kwargs*)
In addition to the parameters below, any keyword arguments supported by TableDataExtractor.TdeTable can be passed in as keyword arguments and they will be passed on to TableDataExtractor.TdeTable.

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its `document` attribute should be set as soon as possible. If the element is being passed in to a *chemdataextractor.doc.document.Document* to initialise it, the `document` attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **caption** (BaseElement) – The caption for the element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **table_data** (*list*) – (Optional) Table data to be passed on to TableDataExtractor to be parsed. Refer to documentation for TableDataExtractor.TdeTable for more information on how this should be structured.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

- **document** (Document) – (Optional) The document containing this element.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.

tde_table = None
TableDataExtractor *TrivialTable* object. Can pass any kwargs into TDE directly.

serialize()
Convert self to a dictionary. The key 'type' will contain the name of the class being serialized, and the key 'caption' will contain a serialized representation of caption, which is a *BaseElement*

definitions
Return a list of all specifier definitions in the caption

Returns list– The specifier definitions

records
All records found in the object, as a list of *BaseModel*.

3.4.6 .doc.text

Text-based document elements.

class chemdataextractor.doc.text.**BaseText**(*text*, *word_tokenizer=None*, *lexicon=None*, *abbreviation_detector=None*, *pos_tagger=None*, *ner_tagger=None*, ***kwargs*)
Bases: *chemdataextractor.doc.element.BaseElement*

Abstract base class for a text Document Element.

__init__(*text*, *word_tokenizer=None*, *lexicon=None*, *abbreviation_detector=None*, *pos_tagger=None*, *ner_tagger=None*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its document attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the document attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words.
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element.
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.

- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

text
The raw text str for this passage of text.

word_tokenizer
The *WordTokenizer* used by this element.

lexicon
The *Lexicon* used by this element.

pos_tagger
The part of speech tagger used by this element. A subclass of *BaseTagger*

ner_tagger
The named entity recognition tagger used by this element. A subclass of *BaseTagger*

tokens
A list of *Token* s for this object.

tags
A list of tags corresponding to each of the tokens in the object. For information on what each of the tags can be, check the documentation on the specific *ner_tagger* and *pos_tagger* used for this class.

definitions
A list of all specifier definitions

chemical_definitions
A list of all chemical label definitiond

serialize()
Convert self to a dictionary. The key 'type' will contain the name of the class being serialized, and the key 'content' will contain a serialized representation of *text*, which is a str

class chemdataextractor.doc.text.**Text**(*text*, *sentence_tokenizer=None*, *word_tokenizer=None*, *lexicon=None*, *abbreviation_detector=None*, *pos_tagger=None*, *ner_tagger=None*, *parsers=None*, ***kwargs*)

Bases: collections.abc.Sequence, *chemdataextractor.doc.text.BaseText*

A passage of text, comprising one or more sentences.

word_tokenizer = <chemdataextractor.nlp.tokenize.ChemWordTokenizer object>

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

abbreviation_detector = <chemdataextractor.nlp.abbrev.ChemAbbreviationDetector object>

pos_tagger = <chemdataextractor.nlp.pos.ChemCrfPosTagger object>

ner_tagger = <chemdataextractor.nlp.cem.CemTagger object>

__init__(*text*, *sentence_tokenizer=None*, *word_tokenizer=None*, *lexicon=None*, *abbreviation_detector=None*, *pos_tagger=None*, *ner_tagger=None*, *parsers=None*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its document attribute should be set as soon as possible. If the element is

being passed in to a *Document* to initialise it, the document attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **sentence_tokenizer** (SentenceTokenizer) – (Optional) Sentence tokenizer for this element. Default *ChemSentenceTokenizer*.
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

sentence_tokenizer = <chemdataextractor.nlp.tokenize.ChemSentenceTokenizer object>

set_config()
Load settings from configuration file

Note: Called when Document instance is created

---

sentences
A list of *Sentence* s that make up this text passage.

raw_sentences
A list of str for the sentences that make up this text passage.

tokens
A list of *Token* s for this object.

raw_tokens
A list of str representations for the tokens of each sentence in this text passage.

pos_tagged_tokens
A list of (*Token* token, str tag) tuples for each sentence in this text passage.

pos_tags
A list of str part of speech tags for each sentence in this text passage.

unprocessed_ner_tagged_tokens
A list of (*Token* token, str named entity recognition tag) from the text.

No corrections from abbreviation detection are performed.

unprocessed_ner_tags
A list of str unprocessed named entity tags for the tokens in this sentence.

No corrections from abbreviation detection are performed.

ner_tagged_tokens
A list of (*Token* token, str named entity recognition tag) from the text.

ner_tags
A list of named entity tags corresponding to each of the tokens in the object. For information on what each of the tags can be, check the documentation on the specific *ner_tagger* used for this object.

cems
A list of all Chemical Entity Mentions in this text as chemdataextractor.doc.text.span

definitions
Return a list of tagged definitions for each sentence in this text passage

chemical_definitions
Return a list of tagged definitions for each sentence in this text passage

tagged_tokens
A list of (*Token* token, str named entity recognition tag) from the text.

tags
A list of tags corresponding to each of the tokens in the object. For information on what each of the tags can be, check the documentation on the specific *ner_tagger* and *pos_tagger* used for this class.

abbreviation_definitions
A list of all abbreviation definitions in this Document. Each abbreviation is in the form (str abbreviation, str long form of abbreviation, str ner_tag)

records
All records found in the object, as a list of *BaseModel*.

class chemdataextractor.doc.text.**Title**(*text*, ***kwargs*)
Bases: *chemdataextractor.doc.text.Text*

__init__(*text*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its document attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the document attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **sentence_tokenizer** (SentenceTokenizer) – (Optional) Sentence tokenizer for this element. Default *ChemSentenceTokenizer*.

- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

class chemdataextractor.doc.text.**Heading**(*text*, ***kwargs*)

Bases: *chemdataextractor.doc.text.Text*

__init__(*text*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its document attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the document attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **sentence_tokenizer** (SentenceTokenizer) – (Optional) Sentence tokenizer for this element. Default *ChemSentenceTokenizer*.
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.

- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

class chemdataextractor.doc.text.**Paragraph**(*text*, ***kwargs*)

Bases: *chemdataextractor.doc.text.Text*

__init__(*text*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its `document` attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the `document` attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **sentence_tokenizer** (SentenceTokenizer) – (Optional) Sentence tokenizer for this element. Default *ChemSentenceTokenizer*.
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

class chemdataextractor.doc.text.**Footnote**(*text*, ***kwargs*)

Bases: *chemdataextractor.doc.text.Text*

__init__(*text*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its `document` attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the `document` attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **sentence_tokenizer** (SentenceTokenizer) – (Optional) Sentence tokenizer for this element. Default *ChemSentenceTokenizer*.
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

class chemdataextractor.doc.text.**Citation**(*text*, *sentence_tokenizer=None*, *word_tokenizer=None*, *lexicon=None*, *abbreviation_detector=None*, *pos_tagger=None*, *ner_tagger=None*, *parsers=None*, ***kwargs*)

Bases: *chemdataextractor.doc.text.Text*

ner_tagger = <chemdataextractor.nlp.tag.NoneTagger object>

No tagging is done for citations

abbreviation_detector = None

class chemdataextractor.doc.text.**Caption**(*text*, ***kwargs*)

Bases: *chemdataextractor.doc.text.Text*

__init__(*text*, ***kwargs*)

Note: If intended as part of a *Document*, an element should either be initialized with a reference to its containing document, or its `document` attribute should be set as soon as possible. If the element is being passed in to a *Document* to initialise it, the `document` attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **sentence_tokenizer** (SentenceTokenizer) – (Optional) Sentence tokenizer for this element. Default *ChemSentenceTokenizer*.
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

definitions

Return a list of tagged definitions for each sentence in this text passage

class chemdataextractor.doc.text.**Sentence**(*text*, *start=0*, *end=None*, *word_tokenizer=None*, *lexicon=None*, *abbreviation_detector=None*, *pos_tagger=None*, *ner_tagger=None*, ***kwargs*)

Bases: *chemdataextractor.doc.text.BaseText*

A single sentence within a text passage.

word_tokenizer = <chemdataextractor.nlp.tokenize.ChemWordTokenizer object>

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

abbreviation_detector = <chemdataextractor.nlp.abbrev.ChemAbbreviationDetector object>

pos_tagger = <chemdataextractor.nlp.pos.ChemCrfPosTagger object>

ner_tagger = <chemdataextractor.nlp.cem.CemTagger object>

__init__(*text*, *start=0*, *end=None*, *word_tokenizer=None*, *lexicon=None*, *abbreviation_detector=None*, *pos_tagger=None*, *ner_tagger=None*, ***kwargs*)

Note: If intended as part of a *chemdataextractor.doc.document.Document*, an element should either be initialized with a reference to its containing document, or its `document` attribute should be set as soon as possible. If the element is being passed in to a *chemdataextractor.doc.document.Document* to initialise it, the `document` attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **start** (*int*) – (Optional) The starting index of the sentence within the containing element. Default 0.
- **end** (*int*) – (Optional) The end index of the sentence within the containing element. Defualt None
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

start = None
The start index of this sentence within the text passage.

end = None
The end index of this sentence within the text passage.

tokens
A list of *Token* s for this object.

raw_tokens
A list of `str` representations for the tokens in the object.

pos_tagged_tokens
A list of (*Token* token, str tag) tuples for each sentence in this sentence.

pos_tags
A list of str part of speech tags for each sentence in this sentence.

unprocessed_ner_tagged_tokens
A list of (*Token* token, str named entity recognition tag) from the text.

No corrections from abbreviation detection are performed.

unprocessed_ner_tags
A list of str unprocessed named entity tags for the tokens in this sentence.

No corrections from abbreviation detection are performed.

abbreviation_definitions
A list of all abbreviation definitions in this Document. Each abbreviation is in the form (str abbreviation, str long form of abbreviation, str ner_tag)

ner_tagged_tokens
A list of (*Token* token, str named entity recognition tag) from the sentence.

ner_tags
A list of named entity tags corresponding to each of the tokens in the object. For information on what each of the tags can be, check the documentation on the specific *ner_tagger* used for this object.

cems
A list of all Chemical Entity Mentions in this text as *Span*

definitions
Return specifier definitions from this sentence

A definition consists of: a) A definition – The quantitity being defined e.g. "Curie Temperature" b) A specifier – The symbol used to define the quantity e.g. "Tc" c) Start – The index of the starting point of the definition d) End – The index of the end point of the definition

Returns list – The specifier definitions

chemical_definitions
Return a list of chemical entity mentions and their associated label

tags
A list of tags corresponding to each of the tokens in the object. For information on what each of the tags can be, check the documentation on the specific *ner_tagger* and *pos_tagger* used for this class.

tagged_tokens
A list of (*Token* token, str named entity recognition tag) from the text.

quantity_re

records
All records found in the object, as a list of *BaseModel*.

class chemdataextractor.doc.text.**Cell**(**args*, ***kwargs*)
Bases: *chemdataextractor.doc.text.Sentence*

Data cell for tables. One row of the category table

__init__(**args*, ***kwargs*)

Note: If intended as part of a *chemdataextractor.doc.document.Document*, an element should either be initialized with a reference to its containing document, or its `document` attribute should be set as soon as possible. If the element is being passed in to a *chemdataextractor.doc.document.Document* to initialise it, the `document` attribute is automatically set during the initialisation of the document, so the user does not need to worry about this.

Parameters

- **text** (*str*) – The text contained in this element.
- **start** (*int*) – (Optional) The starting index of the sentence within the containing element. Default 0.
- **end** (*int*) – (Optional) The end index of the sentence within the containing element. Defualt None
- **word_tokenizer** (WordTokenizer) – (Optional) Word tokenizer for this element. Default *ChemWordTokenizer*.
- **lexicon** (Lexicon) – (Optional) Lexicon for this element. The lexicon stores all the occurences of unique words and can provide Brown clusters for the words. Default *ChemLexicon*
- **abbreviation_detector** (AbbreviationDetector) – (Optional) The abbreviation detector for this element. Default *ChemAbbreviationDetector*.
- **pos_tagger** (BaseTagger) – (Optional) The part of speech tagger for this element. Default *ChemCrfPosTagger*.
- **ner_tagger** (BaseTagger) – (Optional) The named entity recognition tagger for this element. Default *CemTagger*
- **document** (Document) – (Optional) The document containing this element.
- **label** (*str*) – (Optional) The label for the captioned element, e.g. Table 1 would have a label of 1.
- **id** (Any) – (Optional) Some identifier for this element. Must be equatable.
- **models** (*list[chemdataextractor.models.BaseModel]*) – (Optional) A list of models for this element to parse. If the element is part of another element (e.g. a *Sentence* inside a *Paragraph*), or is part of a *Document*, this is set automatically to be the same as that of the containing element, unless manually set otherwise.

classmethod from_tdecell(*tde_cell*, ***kwargs*)

abbreviation_definitions
Empty list. Abbreviation detection is disabled within table cells.

records
Empty list. Individual cells don't provide records, this is handled by the parent Table.

class chemdataextractor.doc.text.**Span**(*text*, *start*, *end*)
Bases: object

A text span within a sentence.

__init__(*text*, *start*, *end*)

Parameters

- **text** (*str*) – The text contained by this span.
- **start** (*int*) – The start offset of this token in the original text.
- **end** (*int*) – The end offsent of this token in the original text.

text = None
The str text content of this span.

start = None
The int start offset of this token in the original text.

end = None
The int end offset of this token in the original text.

length
The int offset length of this span in the original text.

class chemdataextractor.doc.text.**Token**(*text*, *start*, *end*, *lexicon*)
Bases: *chemdataextractor.doc.text.Span*

A single token within a sentence. Corresponds to a word, character, punctuation etc.

__init__(*text*, *start*, *end*, *lexicon*)

Parameters

- **text** (*str*) – The text contained by this token.
- **start** (*int*) – The start offset of this token in the original text.
- **end** (*int*) – The end offsent of this token in the original text.
- **lexicon** (Lexicon) – The lexicon which contains this token.

lexicon = None
The lexicon for this token.

lex
The corresponding *chemdataextractor.nlp.lexicon.Lexeme* entry in the Lexicon for this token.

3.5 .eval

Evaluation of extraction results

3.5.1 .eval.evaluation

3.6 .model

Models for storing relationships extracted using chemdataextractor. The hierarchy for models has been greatly rewritten in 2.0, introducing breaking changes to older scripts using ChemDataExtractor. Please refer to the *migration guide* and the examples for an overview of the changes.

Classes for representing chemical models.

3.6.1 .model.base

Data model for extracted information.

class chemdataextractor.model.base.**BaseType**(*default=None*, *null=False*, *required=False*, *contextual=False*, *parse_expression=None*, *updatable=False*, *binding=False*, *ignore_when_merging=False*)

Bases: object

name = None

__init__(*default=None*, *null=False*, *required=False*, *contextual=False*, *parse_expression=None*, *updatable=False*, *binding=False*, *ignore_when_merging=False*)

Parameters

- **default** – (Optional) The default value for this field if none is set.
- **null** (*bool*) – (Optional) Include in serialized output even if value is None. Default False.
- **required** (*bool*) – (Optional) Whether a value is required. Default False.
- **contextual** (*bool*) – (Optional) Whether this value is contextual. Default False.
- **parse_expression** (BaseParserElement) – (Optional) Expression for parsing, instance of a subclass of BaseParserElement. Default None.
- **updatable** (*bool*) – (Optional) Whether the parse_expression can be changed by the document as parsing occurs. Default False.
- **binding** (*bool*) – (Optional) If this option is set to True, any submodels that have an attribute with the same name must have the same value for this attribute. Default False/
- **ignore_when_merging** (*bool*) – (Optional) If this option is set to True, any records with a different value for this field is treated as corresponding to the same physical record.

reset()
Reset the parse expression to the initial value.

process(*value*)
Convert an assigned value into the desired data format for this field.

serialize(*value*, *primitive=False*)
Serialize this field.

is_empty(*value*)
Return whether a value is considered empty for the case of this field.

class chemdataextractor.model.base.**StringType**(*default=None*, *null=False*, *required=False*, *contextual=False*, *parse_expression=None*, *updatable=False*, *binding=False*, *ignore_when_merging=False*)

Bases: *chemdataextractor.model.base.BaseType*

process(*value*)
Convert value to a unicode string. Useful in case lxml _ElementUnicodeResult are passed from parser.

is_empty(*value*)
Return whether a value is considered empty for the case of this field.

class chemdataextractor.model.base.**FloatType**(*default=None*, *null=False*, *required=False*, *contextual=False*, *parse_expression=None*, *updatable=False*, *binding=False*, *ignore_when_merging=False*)

Bases: *chemdataextractor.model.base.BaseType*

An floating point number field.

process(*value*)
Convert value to a float.

is_empty(*value*)
Return whether a value is considered empty for the case of this field.

class chemdataextractor.model.base.**ModelType**(*model*, ***kwargs*)

Bases: *chemdataextractor.model.base.BaseType*

__init__(*model*, ***kwargs*)

Parameters

- **default** – (Optional) The default value for this field if none is set.
- **null** (*bool*) – (Optional) Include in serialized output even if value is None. Default False.
- **required** (*bool*) – (Optional) Whether a value is required. Default False.
- **contextual** (*bool*) – (Optional) Whether this value is contextual. Default False.
- **parse_expression** (BaseParserElement) – (Optional) Expression for parsing, instance of a subclass of BaseParserElement. Default None.
- **updatable** (*bool*) – (Optional) Whether the parse_expression can be changed by the document as parsing occurs. Default False.
- **binding** (*bool*) – (Optional) If this option is set to True, any submodels that have an attribute with the same name must have the same value for this attribute. Default False/
- **ignore_when_merging** (*bool*) – (Optional) If this option is set to True, any records with a different value for this field is treated as corresponding to the same physical record.

serialize(*value*, *primitive=False*)
Serialize this field.

is_empty(*value*)
Return whether a value is considered empty for the case of this field.

class chemdataextractor.model.base.**ListType**(*field*, *default=None*, *sorted=False*, ***kwargs*)

Bases: *chemdataextractor.model.base.BaseType*

__init__(*field*, *default=None*, *sorted=False*, ***kwargs*)

Parameters

- **default** – (Optional) The default value for this field if none is set.
- **null** (*bool*) – (Optional) Include in serialized output even if value is None. Default False.
- **required** (*bool*) – (Optional) Whether a value is required. Default False.
- **contextual** (*bool*) – (Optional) Whether this value is contextual. Default False.

- **parse_expression** (BaseParserElement) – (Optional) Expression for parsing, instance of a subclass of BaseParserElement. Default None.
- **updatable** (*bool*) – (Optional) Whether the parse_expression can be changed by the document as parsing occurs. Default False.
- **binding** (*bool*) – (Optional) If this option is set to True, any submodels that have an attribute with the same name must have the same value for this attribute. Default False/
- **ignore_when_merging** (*bool*) – (Optional) If this option is set to True, any records with a different value for this field is treated as corresponding to the same physical record.

serialize(*value*, *primitive=False*)
Serialize this field.

is_empty(*value*)
Return whether a value is considered empty for the case of this field.

class chemdataextractor.model.base.**SetType**(*field*, *default=None*, ***kwargs*)
Bases: *chemdataextractor.model.base.BaseType*

__init__(*field*, *default=None*, ***kwargs*)

Parameters

- **default** – (Optional) The default value for this field if none is set.
- **null** (*bool*) – (Optional) Include in serialized output even if value is None. Default False.
- **required** (*bool*) – (Optional) Whether a value is required. Default False.
- **contextual** (*bool*) – (Optional) Whether this value is contextual. Default False.
- **parse_expression** (BaseParserElement) – (Optional) Expression for parsing, instance of a subclass of BaseParserElement. Default None.
- **updatable** (*bool*) – (Optional) Whether the parse_expression can be changed by the document as parsing occurs. Default False.
- **binding** (*bool*) – (Optional) If this option is set to True, any submodels that have an attribute with the same name must have the same value for this attribute. Default False/
- **ignore_when_merging** (*bool*) – (Optional) If this option is set to True, any records with a different value for this field is treated as corresponding to the same physical record.

serialize(*value*, *primitive=False*)
Serialize this field.

is_empty(*value*)
Return whether a value is considered empty for the case of this field.

class chemdataextractor.model.base.**ModelMeta**
Bases: abc.ABCMeta

required_fields

class chemdataextractor.model.base.**BaseModel**(***raw_data*)
Bases: object

fields = {}

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

specifier = None

__init__(***raw_data*)

is_unidentified
If there is no 'compound' field associated with the model but the compound is contextual

classmethod reset_updatables()
Reset all updatable parse_expressions of properties associated with the class.

classmethod update(*definitions*, *strict=True*)
Update this Element's updatable attributes with new information from definitions

Parameters **{list} -- list of definitions found in this element** (*definitions*) –

updated
True/False dependent on if a specifier within the model was updated.

keys()

items()

values()

get(*key*, *default=None*)

contextual_fulfilled
Whether all the contextual fields have been extracted.

Returns True if all fields have been found, False if not.

Return type bool

required_fulfilled
Whether all the required fields have been extracted.

Returns True if all fields have been found, False if not.

Return type bool

noncontextual_required_fulfilled
Whether all the non-contextual required fields have been extracted.

Returns True if all fields have been found, False if not.

Return type bool

serialize(*primitive=False*)
Convert Model to python dictionary.

to_json(**args*, ***kwargs*)
Convert Model to JSON.

is_superset(*other*)
Whether this model instance is a 'superset' of the other model instance.

A model instance is a 'superset' of another if it satisfies the following conditions:

- The model instances are of the same type
- For each of the attributes of the model instances, either:
  - This instance has more information, or
  - Both instances have the same information

Parameters **other** (BaseModel) – The other model instance to compare with this model instance

Returns Whether this model instance is a superset of the other model instance

Return type bool

is_subset(*other*)

Whether this model instance is a 'subset' of the other model instance.

A model instance is a 'subset' of another if it satisfies the following conditions:

- The model instances are of the same type
- For each of the attributes of the model instances, either:
  - The other instance has more information, or
  - Both instances have the same information

Parameters **other** (BaseModel) – The other model instance to compare with this model instance

Returns Whether this model instance is a subset of the other model instance

Return type bool

merge_contextual(*other*)

Merges any fields marked contextual with additional information from other provided that:

- other is of the same type and they don't have any conflicting fields

or

- other is a model type that is part of this model and that field is currently set to be the default value or the field can be merged with the other.

Note: This method mutates the model it's called on **and** returns it.

Parameters **other** (BaseModel) – The other model to merge into this model

Returns A merged model

Return type *BaseModel*

merge_all(*other*)

Merges any properties between other and self, regardless of whether that field is contextual. Checks to make sure that there are no conflicts between the values contained in self and those in other.

Note: This method mutates the model it's called on **and** returns it.

Parameters **other** (BaseModel) – The other model to merge into this model

Returns A merged model

Return type *BaseModel*

classmethod flatten()

A set of all models that are associated with this model. For example, if we have a model like the following with multiple submodels:

```
class A(BaseModel):
    pass

class B(BaseModel):
    a = ModelType(A)

class C(BaseModel):
    b = ModelType(B)
```

then *C.flatten()* would give the result:

```
set(C, B, A)
```

Returns The set of all models associated with this model.

Return type set(*BaseModel*)

binding_properties

A dictionary of all binding properties in this model, and their values.

Note: This function only returns those properties that are immediately binding for this model, and not for any submodels.

Returns A dictionary with the names of all binding fields as the keys and their values as the values.

Return type {str: Any}

record_method

Description (string) of which method was used to create this record.

is_empty

class chemdataextractor.model.base.**ModelList**(**models*)

Bases: collections.abc.MutableSequence

Wrapper around a list of Models objects to facilitate operations on all at once.

__init__(**models*)

Initialize self. See help(type(self)) for accurate signature.

insert(*index*, *value*)

S.insert(index, value) – insert value before index

serialize()

Serialize to a list of python dictionaries.

to_json(**args*, ***kwargs*)

Convert ModelList to JSON.

remove_subsets(*strict=False*)

Remove any subsets contained within the ModelList.

Parameters **strict** (*bool*) – Default True. Whether only strict subsets are removed. When this is False, duplicates are removed too.

3.6.2 .model.model

Model classes for physical properties.

class chemdataextractor.model.model.**Compound**(***raw_data*)
Bases: *chemdataextractor.model.base.BaseModel*

names

labels

roles

parsers = [<chemdataextractor.parse.cem.CompoundParser object>, <chemdataextractor.par

merge(*other*)
Merge data from another Compound into this Compound.

is_unidentified
If there is no 'compound' field associated with the model but the compound is contextual

is_id_only
Return True if identifier information only.

classmethod update(*definitions*, *strict=True*)
Update the Compound labels parse expression

Parameters {list} -- list of definitions found in this element
(*definitions*) –

construct_label_expression(*label*)

fields = {'labels': <chemdataextractor.model.base.SetType object>, 'names': <chemdat

class chemdataextractor.model.model.**Apparatus**(***raw_data*)
Bases: *chemdataextractor.model.base.BaseModel*

name

parsers = [<chemdataextractor.parse.apparatus.ApparatusParser object>]

fields = {'name': <chemdataextractor.model.base.StringType object>}

class chemdataextractor.model.model.**UvvisPeak**(***raw_data*)
Bases: *chemdataextractor.model.base.BaseModel*

value
Peak value, i.e. wavelength

units
Peak value units

extinction

extinction_units

shape

fields = {'extinction': <chemdataextractor.model.base.StringType object>, 'extinction_

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

class chemdataextractor.model.model.**UvvisSpectrum**(***raw_data*)
Bases: *chemdataextractor.model.base.BaseModel*

solvent

temperature

temperature_units

concentration

concentration_units

apparatus

peaks

compound

parsers = [<chemdataextractor.parse.uvvis.UvvisParser object>]

fields = {'apparatus': <chemdataextractor.model.base.ModelType object>, 'compound':

class chemdataextractor.model.model.**IrPeak**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

value

units

strength

bond

fields = {'bond': <chemdataextractor.model.base.StringType object>, 'strength': <che

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

class chemdataextractor.model.model.**IrSpectrum**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

solvent

temperature

temperature_units

concentration

concentration_units

apparatus

peaks

compound

parsers = [<chemdataextractor.parse.ir.IrParser object>]

fields = {'apparatus': <chemdataextractor.model.base.ModelType object>, 'compound':

class chemdataextractor.model.model.**NmrPeak**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

shift

intensity

multiplicity

coupling

coupling_units

number

assignment

fields = {'assignment': <chemdataextractor.model.base.StringType object>, 'coupling':

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

class chemdataextractor.model.model.**NmrSpectrum**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

nucleus

solvent

frequency

frequency_units

standard

temperature

temperature_units

concentration

concentration_units

apparatus

peaks

compound

parsers = [<chemdataextractor.parse.nmr.NmrParser object>]

fields = {'apparatus': <chemdataextractor.model.base.ModelType object>, 'compound':

class chemdataextractor.model.model.**MeltingPoint**(***raw_data*)

Bases: *chemdataextractor.model.units.temperature.TemperatureModel*

solvent

concentration

concentration_units

apparatus

compound

parsers = [<chemdataextractor.parse.mp_new.MpParser object>]

fields = {'apparatus': <chemdataextractor.model.base.ModelType object>, 'compound':

class chemdataextractor.model.model.**GlassTransition**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

A glass transition temperature.

value

units

method

concentration

concentration_units

compound

parsers = [<chemdataextractor.parse.tg.TgParser object>]

fields = {'compound': <chemdataextractor.model.base.ModelType object>, 'concentration

class chemdataextractor.model.model.**QuantumYield**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

A quantum yield measurement.

value

units

solvent

type

standard

standard_value

standard_solvent

concentration

concentration_units

temperature

temperature_units

apparatus

fields = {'apparatus': <chemdataextractor.model.base.ModelType object>, 'concentratio

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

class chemdataextractor.model.model.**FluorescenceLifetime**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

A fluorescence lifetime measurement.

value

units

solvent

concentration

concentration_units

temperature

temperature_units

apparatus

fields = {'apparatus': <chemdataextractor.model.base.ModelType object>, 'concentratio

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

class chemdataextractor.model.model.**ElectrochemicalPotential**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

An oxidation or reduction potential, from cyclic voltammetry.

value

units

type

solvent

concentration

concentration_units

temperature

temperature_units

apparatus

fields = {'apparatus': <chemdataextractor.model.base.ModelType object>, 'concentratio

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

class chemdataextractor.model.model.**NeelTemperature**(***raw_data*)

Bases: *chemdataextractor.model.units.temperature.TemperatureModel*

expression = <chemdataextractor.parse.elements.IWord object>

specifier

compound

fields = {'compound': <chemdataextractor.model.base.ModelType object>, 'error': <che

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.model.**CurieTemperature**(***raw_data*)

Bases: *chemdataextractor.model.units.temperature.TemperatureModel*

expression = <chemdataextractor.parse.elements.First object>

specifier

compound

fields = {'compound': <chemdataextractor.model.base.ModelType object>, 'error': <che

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.model.**InteratomicDistance**(***raw_data*)

Bases: *chemdataextractor.model.units.length.LengthModel*

specifier_expression = <chemdataextractor.parse.elements.And object>

specifier

rij_label = <chemdataextractor.parse.elements.Regex object>

species

compound

another_label

fields = {'another_label': <chemdataextractor.model.base.StringType object>, 'compoun

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.model.**CoordinationNumber**(***raw_data*)

Bases: *chemdataextractor.model.units.quantity_model.DimensionlessModel*

coordination_number_label = <chemdataextractor.parse.elements.Regex object>

specifier_expression = <chemdataextractor.parse.elements.Regex object>

specifier

cn_label

compound

fields = {'cn_label': <chemdataextractor.model.base.StringType object>, 'compound':

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.model.**CNLabel**(***raw_data*)

Bases: *chemdataextractor.model.base.BaseModel*

coordination_number_label = <chemdataextractor.parse.elements.Regex object>

specifier = <chemdataextractor.parse.elements.And object>

label_Juraj

compound

parsers = [<chemdataextractor.parse.auto.AutoSentenceParser object>, <chemdataextracto

fields = {'compound': <chemdataextractor.model.base.ModelType object>, 'label_Juraj':

3.6.3 .model.units

Types for representing quantities, dimensions, and units.

codeauthor Taketomo Isazawa (ti250@cam.ac.uk)

chemdataextractor.model.units.**standard_units**

.model.units.unit

Base types for making units. Refer to the example on *creating new units and dimensions* for more detail on how to create your own units.

class chemdataextractor.model.units.unit.**UnitType**(*default=None*, *null=False*, *required=False*, *contextual=False*, *parse_expression=None*, *updatable=False*, *binding=False*, *ignore_when_merging=False*)

Bases: *chemdataextractor.model.base.BaseType*

A field containing a *Unit* of some type.

process(*value*)

Convert an assigned value into the desired data format for this field.

serialize(*value*, *primitive=False*)

Serialize this field.

is_empty(*value*)

Return whether a value is considered empty for the case of this field.

class chemdataextractor.model.units.unit.**MetaUnit**

Bases: type

Metaclass to ensure that all subclasses of *Unit* take the magnitude into account when converting to standard units.

class chemdataextractor.model.units.unit.**Unit**(*dimensions*, *magnitude=0.0*, *powers=None*)

Bases: object

Object represeting units. Implement subclasses of this for basic units. Units like meters, seconds, and Kelvins are already implemented in ChemDataExtractor. These can then be combined by simply dividing or multiplying them to create more complex units. Alternatively, one can create these by subclassing Unit and setting the powers parameter as desired. For example, a speed could be represented as either:

```
speedunit = Meter() / Second()
```

or

```
class SpeedUnit(Unit):

    def __init__(self, magnitude=1.0):
        super(SpeedUnit, self).__init__(Length()/Time(),
                                        powers={Meter():1.0, Second():-1.0} )

speedunit = SpeedUnit()
```

and either method should produce the same results.

Any subclass of Unit which represents a real unit should implement the following methods:

- convert_value_to_standard
- convert_value_from_standard
- convert_error_to_standard
- convert_error_from_standard

These methods ensure that Units can be seamlessly converted to other ones. Any magnitudes placed in front of the units, e.g. kilometers, are handled automatically. Care must be taken that the 'standard' unit chosen is obvious, consistent, and documented, else another user may implement new units with the same dimensions but a different standard unit, resulting in unexpected errors. To ensure correct behaviour, one should also define the standard unit in code by setting the corresponding dimension's *standard_units*, unless the dimension is a composite one, in which case the standard unit can often be inferred from the constituent units' standard untis

base_magnitude = 0.0

constituent_units = None

Unit instance for showing constituent units. Used for creating more complex models. An example would be:

```
class Newton(Unit):
    constituent_units = Gram(magnitude=3.0) * Meter() * (Second()) ** (-2.0)
```

__init__(*dimensions*, *magnitude=0.0*, *powers=None*)

Creates a unit object. Subclass Unit to create concrete units. For examples, see lengths.py and times.py

Parameters

- **dimensions** (Dimension) – The dimensions this unit is for, e.g. Temperature
- **magnitude** (*float*) – (Optional) The magnitude of the unit. e.g. km would be meters with an magnitude of 3
- **powers** (*dict[Unit : float]*) – (Optional) For representing any more complicated units, e.g. m/s may have this parameter set to {Meter():1.0, Second():-1.0}

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.unit.**DimensionlessUnit**(*magnitude=0.0*)
Bases: *chemdataextractor.model.units.unit.Unit*

Special case to handle dimensionless quantities.

__init__(*magnitude=0.0*)

Parameters **magnitude** (*float*) – The magnitude of the unit.

convert_to_standard(*value*)

convert_error_from_standard(*value*)

convert_error_to_standard(*value*)

convert_from_standard(*value*)

convert_value_from_standard(*value*)

convert_value_to_standard(*value*)

.model.units.dimension

Base types for dimensions. Refer to the example on *creating new units and dimensions* for more detail on how to create your own dimensions.

chemdataextractor.model.units.dimension.**standard_units**

class chemdataextractor.model.units.dimension.**Dimension**
Bases: object

Class for representing physical dimensions.

constituent_dimensions = None
Used for creating composite dimensions. It is of type *Dimension*. An example would be speed, in which case we would have:

```
class Speed(Dimension):
    constituent_dimensions = Length() / Time()
```

units_dict = {}
Used for extracting units with these dimensions. It is of type dictionary{chemdataextractor.parse.element : *Unit* or None}.

An element is the key for None when an element is needed for autoparsing to work correctly, but one does not want to take account of this when extracting a unit from a merged string.

An example of this is °C, which is always split into two tokens, so we need to be able to capture ° and C separately using elements from the *units_dict*, but we do not want this to affect *extract_units()*, to which the single string '°C' is passed in. As a solution, we have the following *units_dict*:

```
units_dict = {R('°?(((K|k)elvin(s)?)|K)\.?', group=0): Kelvin,
      R('(°C|((C|c)elsius))\.?', group=0): Celsius,
      R('°?((F|f)ahrenheit|F)\.?', group=0): Fahrenheit,
      R('°|C', group=0): None}
```

Note: The units_dict has been extensively tested using regex elements, and while in theory it may work with other parse elements, it is strongly recommended that you use a regex element. If a regex element is specified, it should

- Not have a $ symbol at the end: the units can be passed in with numbers or other symbols after it, and these are also used in the autoparser to find candidate tokens which may contain units, and a $ symbol at the end would stop this from working
- Have the group attribute set to 0. Unless this is set, the default behaviour of the regex element is to return the whole token in which the match was found. This is unhelpful behaviour for our logic for extracting units, as we want to extract only the exact characters that matched the unit.

standard_units
The standard units for this dimension. Of type *Unit*.

Set this attribute when creating a new dimension to make converting to the standard units easy via *convert_to_standard()*, and to make it clear in the code what the standard units are.

The standard units when you multiply dimensions together/ have composite dimensions are automatically handled by the class.

class chemdataextractor.model.units.dimension.**Dimensionless**
Bases: *chemdataextractor.model.units.dimension.Dimension*

Special case to handle dimensionless quantities.

standard_units

.model.units.quantity_model

Base types for making quantity models.

codeauthor Taketomo Isazawa (ti250@cam.ac.uk)

class chemdataextractor.model.units.quantity_model.**QuantityModel**(***raw_data*)
Bases: *chemdataextractor.model.base.BaseModel*

Class for modelling quantities. Subclasses of this model can be used in conjunction with Autoparsers to extract properties with zero human intervention. However, they must be constructed in a certain way for them to work optimally with autoparsers. Namely, they should have:

- A specifier field with an associated parse expression (Optional, only required if autoparsers are desired). These parse expressions will be updated automatically using forward-looking Interdependency Resolution if the updatable flag is set to True.
- These specifiers should also have required set to True so that spurious matches are not found.
- If applicable, a compound field, named compound.

Any parse_expressions set in the model should have an added action to ensure that the results are a single word. An example would be to call add_action(join) on each parse expression.

raw_value

raw_units

value

units
A field containing a Unit of some type.

error
An floating point number field.

dimensions = None

specifier

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

convert_to(*unit*)
Convert from current units to the given units. Raises AttributeError if the current unit is not set.

Note: This method both modifies the current model and returns the modified model.

Parameters **unit** (Unit) – The Unit to convert to

Returns The quantity in the given units.

Return type *QuantityModel*

convert_to_standard()
Convert from current units to the standard units. Raises AttributeError if the current unit has not been set or the dimensions do not have standard units.

Note: This method both modifies the current model and returns the modified model.

Returns The quantity in the given units.

Return type *QuantityModel*

convert_value(*from_unit*, *to_unit*)
Convert between the given units. If no units have been set for this model, assumes that it's in standard units.

Parameters

- **from_unit** (Unit) – The Unit to convert from
- **to_unit** (Unit) – The Unit to convert to

Returns The value as expressed in the new unit

Return type float

convert_error(*from_unit*, *to_unit*)
Converts error between given units If no units have been set for this model, assumes that it's in standard units.

Parameters

- **from_unit** (Unit) – The Unit to convert from
- **to_unit** (Unit) – The Unit to convert to

Returns The error as expressed in the new unit

Return type float

is_equal(*other*)
Tests whether the two quantities are physically equal, i.e. whether they represent the same value just in different units.

Parameters **other** (QuantityModel) – The quantity being compared with

Returns Whether the two quantities are equal

Return type bool

is_superset(*other*)
Whether this model instance is a 'superset' of the other model instance.

A model instance is a 'superset' of another if it satisfies the following conditions:

- The model instances are of the same type
- For each of the attributes of the model instances, either:
  - This instance has more information, or
  - Both instances have the same information

Parameters **other** (BaseModel) – The other model instance to compare with this model instance

Returns Whether this model instance is a superset of the other model instance

Return type bool

fields = {'error': <chemdataextractor.model.base.FloatType object>, 'raw_units': <ch

class chemdataextractor.model.units.quantity_model.**DimensionlessModel**(***raw_data*)
Bases: *chemdataextractor.model.units.quantity_model.QuantityModel*

Special case to handle dimensionless quantities

dimensions = <chemdataextractor.model.units.dimension.Dimensionless object>

raw_units

fields = {'error': <chemdataextractor.model.base.FloatType object>, 'raw_units': <ch

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

.model.units.length

Units and models for lengths.

codeauthor Taketomo Isazawa (ti250@cam.ac.uk)

class chemdataextractor.model.units.length.**Length**
Bases: *chemdataextractor.model.units.dimension.Dimension*

Dimension subclass for lengths.

standard_units

units_dict = {<chemdataextractor.parse.elements.Regex object>: <class 'chemdataextrac

class chemdataextractor.model.units.length.**LengthModel**(***raw_data*)
Bases: *chemdataextractor.model.units.quantity_model.QuantityModel*

Model for lengths.

dimensions = <chemdataextractor.model.units.length.Length object>

fields = {'error': <chemdataextractor.model.base.FloatType object>, 'raw_units': <ch

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.units.length.**LengthUnit**(*magnitude=0.0*, *powers=None*)

Bases: *chemdataextractor.model.units.unit.Unit*

Base class for units with dimensions of length. The standard value for length is defined to be a meter, implemented in the Meter class.

__init__(*magnitude=0.0*, *powers=None*)

Creates a unit object. Subclass Unit to create concrete units. For examples, see lengths.py and times.py

Parameters

- **dimensions** (Dimension) – The dimensions this unit is for, e.g. Temperature
- **magnitude** (*float*) – (Optional) The magnitude of the unit. e.g. km would be meters with an magnitude of 3
- **powers** (*dict[Unit : float]*) – (Optional) For representing any more complicated units, e.g. m/s may have this parameter set to {Meter():1.0, Second():-1.0}

convert_error_from_standard(*value*)

convert_error_to_standard(*value*)

convert_value_from_standard(*value*)

convert_value_to_standard(*value*)

class chemdataextractor.model.units.length.**Meter**(*magnitude=0.0*, *powers=None*)

Bases: *chemdataextractor.model.units.length.LengthUnit*

Class for meters.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.length.**Mile**(*magnitude=0.0*, *powers=None*)

Bases: *chemdataextractor.model.units.length.LengthUnit*

Class for miles.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.length.**Angstrom**(*magnitude=0.0*, *powers=None*)

Bases: *chemdataextractor.model.units.length.LengthUnit*

Class for Angstroms.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.length.**Micron**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.length.LengthUnit*

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

.model.units.mass

Units and models for masses.

codeauthor Taketomo Isazawa (ti250@cam.ac.uk)

class chemdataextractor.model.units.mass.**Mass**
Bases: *chemdataextractor.model.units.dimension.Dimension*

Dimension subclass for masses.

standard_units

units_dict = {<chemdataextractor.parse.elements.Regex object>: <class 'chemdataextrac

class chemdataextractor.model.units.mass.**MassModel**(***raw_data*)
Bases: *chemdataextractor.model.units.quantity_model.QuantityModel*

Model for mass.

dimensions = <chemdataextractor.model.units.mass.Mass object>

fields = {'error': <chemdataextractor.model.base.FloatType object>, 'raw_units': <ch

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.units.mass.**MassUnit**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.unit.Unit*

Base class for units with dimensions of mass. The standard value for mass is defined to be a kilogram, which can be created with Gram(magnitude=3.0)

__init__(*magnitude=0.0*, *powers=None*)
Creates a unit object. Subclass Unit to create concrete units. For examples, see lengths.py and times.py

Parameters

- **dimensions** (Dimension) – The dimensions this unit is for, e.g. Temperature
- **magnitude** (*float*) – (Optional) The magnitude of the unit. e.g. km would be meters with an magnitude of 3
- **powers** (*dict[Unit : float]*) – (Optional) For representing any more complicated units, e.g. m/s may have this parameter set to {Meter():1.0, Second():-1.0}

convert_error_from_standard(*value*)

convert_error_to_standard(*value*)

convert_value_from_standard(*value*)

convert_value_to_standard(*value*)

class chemdataextractor.model.units.mass.**Gram**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.mass.MassUnit*

Class for grams.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.mass.**Pound**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.mass.MassUnit*

Class for pounds.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.mass.**Tonne**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.mass.MassUnit*

Class for tonnes, i.e. metric tons.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

.model.units.time

Units and models for times.

codeauthor Taketomo Isazawa (ti250@cam.ac.uk)

class chemdataextractor.model.units.time.**Time**
Bases: *chemdataextractor.model.units.dimension.Dimension*

Dimension subclass for times.

standard_units

units_dict = {<chemdataextractor.parse.elements.Regex object>: <class 'chemdataextrac

class chemdataextractor.model.units.time.**TimeModel**(***raw_data*)
Bases: *chemdataextractor.model.units.quantity_model.QuantityModel*

Model for times. These models should strictly be used for time intervals, never absolute times, as peculiarities of calendars are not supported, e.g. a minute is always defined as 60 seconds.

dimensions = <chemdataextractor.model.units.time.Time object>

fields = {'error': <chemdataextractor.model.base.FloatType object>, 'raw_units': <ch

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.units.time.**TimeUnit**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.unit.Unit*

__init__(*magnitude=0.0*, *powers=None*)
Base class for units with dimensions of time. The standard value for time is defined to be a second, implemented in the Second class.

convert_error_from_standard(*value*)

convert_error_to_standard(*value*)

convert_value_from_standard(*value*)

convert_value_to_standard(*value*)

class chemdataextractor.model.units.time.**Second**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.time.TimeUnit*

Class for seconds.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.time.**Hour**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.time.TimeUnit*

Class for hours.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.time.**Minute**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.time.TimeUnit*

Class for minutes.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.time.**Year**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.time.TimeUnit*

Class for years.

convert_to_standard(*value*)

convert_from_standard(*value*)

convert_error_from_standard(*value*)

convert_error_to_standard(*value*)

convert_value_from_standard(*value*)

convert_value_to_standard(*value*)

class chemdataextractor.model.units.time.**Day**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.time.TimeUnit*

Class for days.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

.model.units.temperature

Units and models for temperatures.

codeauthor Taketomo Isazawa (ti250@cam.ac.uk)

class chemdataextractor.model.units.temperature.**Temperature**
Bases: *chemdataextractor.model.units.dimension.Dimension*

Dimension subclass for temperatures.

standard_units

units_dict = {<chemdataextractor.parse.elements.Regex object>: <class 'chemdataextrac

class chemdataextractor.model.units.temperature.**TemperatureModel**(***raw_data*)
Bases: *chemdataextractor.model.units.quantity_model.QuantityModel*

Model for temperatures.

dimensions = <chemdataextractor.model.units.temperature.Temperature object>

fields = {'error': <chemdataextractor.model.base.FloatType object>, 'raw_units': <ch

parsers = [<chemdataextractor.parse.template.MultiQuantityModelTemplateParser object>,

class chemdataextractor.model.units.temperature.**TemperatureUnit**(*magnitude=0.0*, *powers=None*)
Bases: *chemdataextractor.model.units.unit.Unit*

Base class for units with dimensions of temperature. The standard value for temperature is defined to be a Kelvin, implemented in the Kelvin class.

__init__(*magnitude=0.0*, *powers=None*)
Creates a unit object. Subclass Unit to create concrete units. For examples, see lengths.py and times.py

Parameters

- **dimensions** (Dimension) – The dimensions this unit is for, e.g. Temperature
- **magnitude** (*float*) – (Optional) The magnitude of the unit. e.g. km would be meters with an magnitude of 3
- **powers** (*dict[Unit : float]*) – (Optional) For representing any more complicated units, e.g. m/s may have this parameter set to {Meter():1.0, Second():-1.0}

convert_error_from_standard(*value*)

convert_error_to_standard(*value*)

convert_value_from_standard(*value*)

convert_value_to_standard(*value*)

class chemdataextractor.model.units.temperature.**Kelvin**(*magnitude=0.0*, *powers=None*)

Bases: *chemdataextractor.model.units.temperature.TemperatureUnit*

Class for Kelvins.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.temperature.**Celsius**(*magnitude=0.0*, *powers=None*)

Bases: *chemdataextractor.model.units.temperature.TemperatureUnit*

Class for Celsius

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

class chemdataextractor.model.units.temperature.**Fahrenheit**(*magnitude=0.0*, *powers=None*)

Bases: *chemdataextractor.model.units.temperature.TemperatureUnit*

Class for Fahrenheit.

convert_value_to_standard(*value*)

convert_value_from_standard(*value*)

convert_error_to_standard(*value*)

convert_error_from_standard(*value*)

3.7 .nlp

Tools for performing the NLP stages, such as POS tagging, Word clustering, CNER, Abbreviation detection

Chemistry-aware natural language processing framework.

3.7.1 .nlp.abbrev

Abbreviation detection.

class chemdataextractor.nlp.abbrev.**AbbreviationDetector**(*abbr_min=None*, *abbr_max=None*, *abbr_equivs=None*)

Bases: `object`

Detect abbreviation definitions in a list of tokens.

Similar to the algorithm in Schwartz & Hearst 2003.

__init__ (*abbr_min=None*, *abbr_max=None*, *abbr_equivs=None*)
Initialize self. See help(type(self)) for accurate signature.

abbr_min = 3
Minimum abbreviation length

abbr_max = 10
Maximum abbreviation length

abbr_equivs = []
String equivalents to use when detecting abbreviations.

detect (*tokens*)
Return a (abbr, long) pair for each abbreviation definition.

detect_spans (*tokens*)
Return (abbr_span, long_span) pair for each abbreviation definition.

abbr_span and long_span are (int, int) spans defining token ranges.

class chemdataextractor.nlp.abbrev.**ChemAbbreviationDetector**(*abbr_min=None*, *abbr_max=None*, *abbr_equivs=None*)
Bases: *chemdataextractor.nlp.abbrev.AbbreviationDetector*

Chemistry-aware abbreviation detector.

This abbreviation detector has an additional list of string equivalents (e.g. Silver = Ag) that improve abbreviation detection on chemistry texts.

abbr_min = 3
Minimum abbreviation length

abbr_max = 10
Maximum abbreviation length

abbr_equivs = [('silver', 'Ag'), ('gold', 'Au'), ('mercury', 'Hg'), ('lead', 'Pb'), ('
String equivalents to use when detecting abbreviations.

3.7.2 .nlp.cem

Named entity recognition (NER) for Chemical entity mentions (CEM).

chemdataextractor.nlp.cem.**IGNORE_SUFFIX = ['-', "'s", '-activated', '-adequate', '-affected**
Token endings to ignore when considering stopwords and deriving spans

chemdataextractor.nlp.cem.**IGNORE_PREFIX = ['fluorophore-', 'low-', 'high-', 'single-', 'odd**
Token beginnings to ignore when considering stopwords and deriving spans

chemdataextractor.nlp.cem.**STRIP_END = ['groups', 'group', 'colloidal', 'dyes', 'dye', 'prod**
Final tokens to remove from entity matches

chemdataextractor.nlp.cem.**STRIP_START = ['anhydrous', 'elemental', 'amorphous', 'conjugated**
First tokens to remove from entity matches

chemdataextractor.nlp.cem.**STOP_TOKENS = {'.cdx', '.sk2', '10.1021', '10.1039', '10.1186',**
Disallowed tokens in chemical entity mentions (discard if any single token has exact case-insensitive match)

chemdataextractor.nlp.cem.**STOP_SUB = {' brand of ', ' oil', ' with ', '!', '%', ', ', ';',**
Disallowed substrings in chemical entity mentions (only used when filtering to construct the dictionary?)

chemdataextractor.nlp.cem.**STOPLIST = {'(gaba)ergic', '1,3-dpma', '1,5-dpma', '12mg', '3 ps**

chemdataextractor.nlp.cem.**STOP_RES = ['^(http|ftp)://', '\\.(com|uk|eu|org|net)$', '^\\d{4**

the entity text is passed as lowercase.

Type Regular expressions that define disallowed chemical entity mentions. Note

chemdataextractor.nlp.cem.**SPLITS = ['^(actinium|aluminium|aluminum|americium|antimony|argon**

Regular expressions defining collections of words that should be split if joined by hyphens or -to-

class chemdataextractor.nlp.cem.**CiDictCemTagger**(*words=None*, *model=None*, *entity=None*, *case_sensitive=None*, *lexicon=None*)

Bases: *chemdataextractor.nlp.tag.DictionaryTagger*

Case-insensitive CEM dictionary tagger.

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

model = 'models/cem_dict-1.0.pickle'

class chemdataextractor.nlp.cem.**CsDictCemTagger**(*words=None*, *model=None*, *entity=None*, *case_sensitive=None*, *lexicon=None*)

Bases: *chemdataextractor.nlp.tag.DictionaryTagger*

Case-sensitive CEM dictionary tagger.

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

model = 'models/cem_dict_cs-1.0.pickle'

case_sensitive = True

class chemdataextractor.nlp.cem.**CrfCemTagger**(*model=None*, *lexicon=None*, *clusters=None*, *params=None*)

Bases: *chemdataextractor.nlp.tag.CrfTagger*

model = 'models/cem_crf_chemdner_cemp-1.0.pickle'

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

clusters = True

params = {'c1': 1.0, 'c2': 0.001, 'feature.possible_states': False, 'feature.possib

class chemdataextractor.nlp.cem.**CemTagger**

Bases: *chemdataextractor.nlp.tag.BaseTagger*

Return the combined output of a number of chemical entity taggers.

taggers = [<chemdataextractor.nlp.cem.CrfCemTagger object>, <chemdataextractor.nlp.cem

The individual chemical entity taggers to use.

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

tag(*tokens*)

Run individual chemical entity mention taggers and return union of matches, with some postprocessing.

3.7.3 .nlp.corpus

Tools for reading and writing text corpora.

class chemdataextractor.nlp.corpus.**LazyCorpusLoader**(*name*, *reader_cls*, **args*, ***kwargs*)

Bases: object

Derived from NLTK LazyCorpusLoader.

__init__(*name*, *reader_cls*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

chemdataextractor.nlp.corpus.**wsj = <BracketParseCorpusReader in '.../corpora/wsj_training'**
Penn Treebank Revised, LDC2015T13)

Type Entire WSJ corpus (English News Text Treebank

chemdataextractor.nlp.corpus.**wsj_training = <BracketParseCorpusReader in '.../corpora/wsj_t**
Penn Treebank Revised, LDC2015T13)

Type WSJ corpus sections 0-18 (English News Text Treebank

chemdataextractor.nlp.corpus.**wsj_development = <BracketParseCorpusReader in '.../corpora/ws**
Penn Treebank Revised, LDC2015T13)

Type WSJ corpus sections 19-21 (English News Text Treebank

chemdataextractor.nlp.corpus.**wsj_evaluation = <BracketParseCorpusReader in '.../corpora/ws**
Penn Treebank Revised, LDC2015T13)

Type WSJ corpus sections 22-24 (English News Text Treebank

chemdataextractor.nlp.corpus.**treebank2_training = <ChunkedCorpusReader in '.../corpora/tree**
WSJ corpus sections 0-18 (treebank2)

chemdataextractor.nlp.corpus.**treebank2_development = <ChunkedCorpusReader in '.../corpora/t**
WSJ corpus sections 19-21 (treebank2)

chemdataextractor.nlp.corpus.**treebank2_evaluation = <ChunkedCorpusReader in '.../corpora/tr**
WSJ corpus sections 22-24 (treebank2)

chemdataextractor.nlp.corpus.**genia_training = <TaggedCorpusReader in '.../corpora/genia_tra**
First 80% of GENIA POS-tagged corpus

chemdataextractor.nlp.corpus.**genia_evaluation = <TaggedCorpusReader in '.../corpora/genia_e**
Last 20% of GENIA POS-tagged corpus

chemdataextractor.nlp.corpus.**medpost = <TaggedCorpusReader in '.../corpora/medpost' (not lo**

chemdataextractor.nlp.corpus.**medpost_training = <TaggedCorpusReader in '.../corpora/medpost**

chemdataextractor.nlp.corpus.**medpost_evaluation = <TaggedCorpusReader in '.../corpora/medpo**

chemdataextractor.nlp.corpus.**cde_tokensc = <PlaintextCorpusReader in '.../corpora/cde_token**

chemdataextractor.nlp.corpus.**chemdner_training = <PlaintextCorpusReader in '.../corpora/che**

3.7.4 .nlp.lexicon

Cache features of previously seen words.

class chemdataextractor.nlp.lexicon.**Lexeme**(*text*, *normalized*, *lower*, *first*, *suffix*, *shape*, *length*, *upper_count*, *lower_count*, *digit_count*, *is_alpha*, *is_ascii*, *is_digit*, *is_lower*, *is_upper*, *is_title*, *is_punct*, *is_hyphenated*, *like_url*, *like_number*, *cluster*)

Bases: object

__init__(*text*, *normalized*, *lower*, *first*, *suffix*, *shape*, *length*, *upper_count*, *lower_count*, *digit_count*, *is_alpha*, *is_ascii*, *is_digit*, *is_lower*, *is_upper*, *is_title*, *is_punct*, *is_hyphenated*, *like_url*, *like_number*, *cluster*)
Initialize self. See help(type(self)) for accurate signature.

text
Original Lexeme text.

cluster
The Brown Word Cluster for this Lexeme.

normalized
Normalized text, using the Lexicon Normalizer.

lower
Lowercase text.

first
First character.

suffix
Three-character suffix

shape
Word shape. Derived by replacing every number with 'd', every greek letter with 'g', and every latin letter with 'X' or 'x' for uppercase and lowercase respectively.

length
Lexeme length.

upper_count
Count of uppercase characters.

lower_count
Count of lowercase characters.

digit_count
Count of digits.

is_alpha
Whether the text is entirely alphabetical characters.

is_ascii
Whether the text is entirely ASCII characters.

is_digit
Whether the text is entirely digits.

is_lower
Whether the text is entirely lowercase.

is_upper
Whether the text is entirely uppercase.

is_title
Whether the text is title cased.

is_punct
Whether the text is entirely punctuation characters.

is_hyphenated
Whether the text is hyphenated.

like_url
Whether the text looks like a URL.

like_number
Whether the text looks like a number.

class chemdataextractor.nlp.lexicon.**Lexicon**

Bases: object

normalizer = <chemdataextractor.text.normalize.Normalizer object>

The Normalizer for this Lexicon.

clusters_path = None

Path to the Brown clusters model file for this Lexicon.

__init__()

add(*text*)

Add text to the lexicon.

Parameters **text** (*string*) – The text to add.

cluster(*text*)

normalized(*text*)

lower(*text*)

first(*text*)

suffix(*text*)

shape(*text*)

length(*text*)

digit_count(*text*)

upper_count(*text*)

lower_count(*text*)

is_alpha(*text*)

is_ascii(*text*)

is_digit(*text*)

is_lower(*text*)

is_upper(*text*)

is_title(*text*)

is_punct(*text*)

is_hyphenated(*text*)

like_url(*text*)

like_number(*text*)

class chemdataextractor.nlp.lexicon.**ChemLexicon**

Bases: *chemdataextractor.nlp.lexicon.Lexicon*

A Lexicon that is pre-configured with a Chemistry-aware Normalizer and Brown word clusters derived from a chemistry corpus.

normalizer = <chemdataextractor.text.normalize.ChemNormalizer object>

clusters_path = 'models/clusters_chem1500-1.0.pickle'

3.7.5 .nlp.pos

Part-of-speech tagging.

chemdataextractor.nlp.pos.**TAGS = ['NN', 'IN', 'NNP', 'DT', 'NNS', 'JJ', ',', '.', 'CD', 'RB']**
Complete set of POS tags. Ordered by decreasing frequency in WSJ corpus.

class chemdataextractor.nlp.pos.**ApPosTagger**(*model=None*, *lexicon=None*, *clusters=None*)
Bases: *chemdataextractor.nlp.tag.ApTagger*

Greedy Averaged Perceptron POS tagger trained on WSJ corpus.

model = 'models/pos_ap_wsj_nocluster-1.0.pickle'

clusters = False

class chemdataextractor.nlp.pos.**ChemApPosTagger**(*model=None*, *lexicon=None*, *clusters=None*)
Bases: *chemdataextractor.nlp.pos.ApPosTagger*

Greedy Averaged Perceptron POS tagger trained on both WSJ and GENIA corpora.

Uses features based on word clusters from chemistry text.

model = 'models/pos_ap_wsj_genia-1.0.pickle'

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

clusters = True

class chemdataextractor.nlp.pos.**CrfPosTagger**(*model=None*, *lexicon=None*, *clusters=None*, *params=None*)
Bases: *chemdataextractor.nlp.tag.CrfTagger*

model = 'models/pos_crf_wsj_nocluster-1.0.pickle'

clusters = False

class chemdataextractor.nlp.pos.**ChemCrfPosTagger**(*model=None*, *lexicon=None*, *clusters=None*, *params=None*)
Bases: *chemdataextractor.nlp.pos.CrfPosTagger*

model = 'models/pos_crf_wsj_genia-1.0.pickle'

lexicon = <chemdataextractor.nlp.lexicon.ChemLexicon object>

clusters = True

3.7.6 .nlp.tag

Tagger implementations. Used for part-of-speech tagging and named entity recognition.

class chemdataextractor.nlp.tag.**BaseTagger**
Bases: object

Abstract tagger class from which all taggers inherit.

Subclasses must implement a `tag()` method.

tag(*tokens*)
Return a list of (token, tag) tuples for the given list of token strings.

Parameters **tokens** (*list(str)*) – The list of tokens to tag.

Return type list(tuple(str, str))

tag_sents(*sentences*)
Apply the `tag` method to each sentence in `sentences`.

evaluate(*gold*)
Evaluate the accuracy of this tagger using a gold standard corpus.

Parameters **str))) gold** (*list(list(tuple(str,)*) – The list of tagged sentences to score the tagger on.

Returns Tagger accuracy value.

Return type float

class chemdataextractor.nlp.tag.**NoneTagger**
Bases: *chemdataextractor.nlp.tag.BaseTagger*

Tag every token with None.

tag(*tokens*)

class chemdataextractor.nlp.tag.**RegexTagger**(*patterns=None*, *lexicon=None*)
Bases: *chemdataextractor.nlp.tag.BaseTagger*

Regular Expression Tagger.

__init__(*patterns=None*, *lexicon=None*)

Parameters **string)) patterns** (*list(tuple(string,)*) – List of (regex, tag) pairs.

patterns = [('^-?[0-9]+(.[0-9]+)?$', 'CD'), ('(The|the|A|a|An|an)$', 'AT'), ('.*able$'
Regular expression patterns in (regex, tag) tuples.

lexicon = <chemdataextractor.nlp.lexicon.Lexicon object>
The lexicon to use

tag(*tokens*)
Return a list of (token, tag) tuples for a given list of tokens.

class chemdataextractor.nlp.tag.**AveragedPerceptron**
Bases: object

Averaged Perceptron implementation.

Based on implementation by Matthew Honnibal, released under the MIT license.

See more: http://spacy.io/blog/part-of-speech-POS-tagger-in-python/ https://github.com/sloria/textblob-aptagger

__init__()
Initialize self. See help(type(self)) for accurate signature.

predict(*features*)
Dot-product the features and current weights and return the best label.

update(*truth*, *guess*, *features*)
Update the feature weights.

average_weights()
Average weights from all iterations.

save(*path*)
Save the pickled model weights.

load(*path*)
Load the pickled model weights.

class chemdataextractor.nlp.tag.**ApTagger**(*model=None*, *lexicon=None*, *clusters=None*)

Bases: *chemdataextractor.nlp.tag.BaseTagger*

Greedy Averaged Perceptron tagger, based on implementation by Matthew Honnibal, released under the MIT license.

See more: http://spacy.io/blog/part-of-speech-POS-tagger-in-python/ https://github.com/sloria/textblob-aptagger

START = ['-START-', '-START2-']

__init__(*model=None*, *lexicon=None*, *clusters=None*)

lexicon = <chemdataextractor.nlp.lexicon.Lexicon object>

clusters = False

tag(*tokens*)

Return a list of (token, tag) tuples for a given list of tokens.

train(*sentences*, *nr_iter=5*)

Train a model from sentences.

Parameters

- **sentences** – A list of sentences, each of which is a list of (token, tag) tuples.
- **nr_iter** – Number of training iterations.

save(*f*)

Save pickled model to file.

load(*model*)

Load pickled model.

class chemdataextractor.nlp.tag.**CrfTagger**(*model=None*, *lexicon=None*, *clusters=None*, *params=None*)

Bases: *chemdataextractor.nlp.tag.BaseTagger*

Tagger that uses Conditional Random Fields (CRF).

__init__(*model=None*, *lexicon=None*, *clusters=None*, *params=None*)

lexicon = <chemdataextractor.nlp.lexicon.Lexicon object>

clusters = False

params = {'c1': 1.0, 'c2': 0.001, 'feature.possible_states': False, 'feature.possib

//www.chokkan.org/software/crfsuite/manual.html

Type Parameters to pass to training algorithm. See http

load(*model*)

tag(*tokens*)

Return a list of ((token, tag), label) tuples for a given list of (token, tag) tuples.

train(*sentences*, *model*)

Train the CRF tagger using CRFSuite.

Params sentences Annotated sentences.

Params model Path to save pickled model.

class chemdataextractor.nlp.tag.**DictionaryTagger**(*words=None*, *model=None*, *entity=None*, *case_sensitive=None*, *lexicon=None*)

Bases: *chemdataextractor.nlp.tag.BaseTagger*

Dictionary Tagger. Tag tokens based on inclusion in a DAWG.

delimiters = re.compile('(^.|\\b|\\s|\\W|.$)')
Delimiters that define where matches are allowed to start or end.

__init__(*words=None*, *model=None*, *entity=None*, *case_sensitive=None*, *lexicon=None*)

Parameters **words** (*list(list(string))*) – list of words, each of which is a list of tokens.

model = None
DAWG model file path.

entity = 'CM'
Optional no B/I?

Type Entity tag. Matches will be tagged like 'B-CM' and 'I-CM' according to IOB scheme. TODO

case_sensitive = False
Whether dictionary matches are case sensitive.

lexicon = <chemdataextractor.nlp.lexicon.Lexicon object>
The lexicon to use.

load(*model*)
Load pickled DAWG from disk.

save(*path*)
Save pickled DAWG to disk.

build(*words*)
Construct dictionary DAWG from tokenized words.

tag(*tokens*)
Return a list of (token, tag) tuples for a given list of tokens.

3.7.7 .nlp.tokenize

Word and sentence tokenizers.

class chemdataextractor.nlp.tokenize.**BaseTokenizer**
Bases: object

Abstract base class from which all Tokenizer classes inherit.

Subclasses must implement a `span_tokenize(text)` method that returns a list of integer offset tuples that identify tokens in the text.

tokenize(*s*)
Return a list of token strings from the given sentence.

Parameters **s** (*string*) – The sentence string to tokenize.

Return type iter(str)

span_tokenize(*s*)
Return a list of integer offsets that identify tokens in the given sentence.

Parameters **s** (*string*) – The sentence string to tokenize.

Return type iter(tuple(int, int))

chemdataextractor.nlp.tokenize.**regex_span_tokenize**(*s*, *regex*)
Return spans that identify tokens in s split using regex.

class chemdataextractor.nlp.tokenize.**SentenceTokenizer**(*model=None*)
Bases: *chemdataextractor.nlp.tokenize.BaseTokenizer*

Sentence tokenizer that uses the Punkt algorithm by Kiss & Strunk (2006).

__init__(*model=None*)
Initialize self. See help(type(self)) for accurate signature.

model = 'models/punkt_english.pickle'

get_sentences(*text*)

span_tokenize(*s*)
Return a list of integer offsets that identify sentences in the given text.

Parameters **s** (*string*) – The text to tokenize into sentences.

Return type iter(tuple(int, int))

class chemdataextractor.nlp.tokenize.**ChemSentenceTokenizer**(*model=None*)
Bases: *chemdataextractor.nlp.tokenize.SentenceTokenizer*

Sentence tokenizer that uses the Punkt algorithm by Kiss & Strunk (2006), trained on chemistry text.

model = 'models/punkt_chem-1.0.pickle'

class chemdataextractor.nlp.tokenize.**WordTokenizer**(*split_last_stop=True*)
Bases: *chemdataextractor.nlp.tokenize.BaseTokenizer*

Standard word tokenizer for generic English text.

SPLIT = ['----', '----', '--------', '<--->', '---', '---', '------', '<-->', '-->', '
Split before and after these sequences, wherever they occur, unless entire token is one of these sequences

SPLIT_NO_DIGIT = [':', ',']
Split around these sequences unless they are followed by a digit

SPLIT_START_WORD = ["''", '``', "'"]
Split after these sequences if they start a word

SPLIT_END_WORD = ["'s", "'m", "'d", "'ll", "'re", "'ve", "n't", "''", "'", '’s', '’m',
Split before these sequences if they end a word

NO_SPLIT_STOP = ['...', 'al.', 'Co.', 'Ltd.', 'Pvt.', 'A.D.', 'B.C.', 'B.V.', 'S.D.',
Don’t split full stop off last token if it is one of these sequences

CONTRACTIONS = [('cannot', 3), ("d'ye", 1), ('d’ye', 1), ('gimme', 3), ('gonna', 3), (
Split these contractions at the specified index

NO_SPLIT = {'mm-hm', 'mm-mm', 'o-kay', 'uh-huh', 'uh-oh', 'wanna-be'}
Don’t split these sequences.

NO_SPLIT_PREFIX = {'a', 'agro', 'ante', 'anti', 'arch', 'be', 'bi', 'bio', 'co', 'coun
Don’t split around hyphens with these prefixes

NO_SPLIT_SUFFIX = {'-o-torium', 'esque', 'ette', 'fest', 'fold', 'gate', 'itis', 'less
Don’t split around hyphens with these suffixes.

NO_SPLIT_CHARS = '0123456789,\'"“”„‘’,`´'
Don't split around hyphens if only these characters before or after.

__init__(*split_last_stop=True*)
Initialize self. See help(type(self)) for accurate signature.

split_last_stop = None
Whether to split off the final full stop (unless preceded by NO_SPLIT_STOP). Default True.

get_word_tokens(*sentence*, *additional_regex=None*)

get_additional_regex(*sentence*)

handle_additional_regex(*s*, *span*, *nextspan*, *additional_regex*)

span_tokenize(*s*, *additional_regex=None*)

class chemdataextractor.nlp.tokenize.**ChemWordTokenizer**(*split_last_stop=True*)
Bases: *chemdataextractor.nlp.tokenize.WordTokenizer*

Word Tokenizer for chemistry text.

SPLIT = ['----', '----', '--------', '<--->', '---', '---', '------', '<-->', '-->', '
Split before and after these sequences, wherever they occur, unless entire token is one of these sequences

SPLIT_END = [':', ',', '(TM)', '(R)', '(®)', '(™)', '()', '()', '()', '()', '()', '(
Split before these sequences if they end a token

SPLIT_END_NO_DIGIT = ['(aq)', '(aq.)', '(s)', '(l)', '(g)']
Split before these sequences if they end a token, unless preceded by a digit

NO_SPLIT_SLASH = ['+', '-', '']
Don't split around slash when both preceded and followed by these characters

QUANTITY_RE = re.compile('^((?P<split>\\d\\d\\d)g|(?P<_split1>[-]?\\d+\\.\\d+|10[-]\\d
Regular expression that matches a numeric quantity with units

NO_SPLIT_PREFIX_ENDING = re.compile('(^\\(.*\\)|^[\\d,\'"“”„‘’,`´A-Ωα-ω]+|ano|ato|azo|
Don't split on hyphen if the prefix matches this regular expression

NO_SPLIT_CHEM = re.compile('([\\-α-ω]|\\d+,\\d+|\\d+[A-Z]|^d\\d\\d?$|acetic|acetyl|aci
Don't split on hyphen if prefix or suffix match this regular expression

NO_SPLIT_PREFIX = {'a', 'aci', 'adeno', 'agro', 'aldehydo', 'allo', 'alpha', 'altro',
Don't split on hyphen if the prefix is one of these sequences

SPLIT_SUFFIX = {'absorption', 'abstinent', 'abstraction', 'abuse', 'accelerated', 'acc
Split on hyphens followed by one of these sequences

NO_SPLIT = {'°c'}

get_additional_regex(*sentence*)

class chemdataextractor.nlp.tokenize.**FineWordTokenizer**(*split_last_stop=True*)
Bases: *chemdataextractor.nlp.tokenize.WordTokenizer*

Word Tokenizer that also split around hyphens and all colons.

SPLIT = ['----', '----', '--------', '<--->', '---', '---', '------', '<-->', '-->', '
Split before and after these sequences, wherever they occur, unless entire token is one of these sequences

SPLIT_NO_DIGIT = [',']
Split before these sequences if they end a token

NO_SPLIT = {}

NO_SPLIT_PREFIX = {}
Don't split around hyphens with these prefixes

NO_SPLIT_SUFFIX = {}
Don't split around hyphens with these suffixes.

3.8 .parse

Chemical property parsers. Parsers have been refactored in 2.0 which has introduced breaking changes to older code. Please refer to the examples and the *migration guide* for 2.0 for an overview of the changes.

Parse text using rule-based grammars.

3.8.1 .parse.actions

Actions to perform during parsing.

chemdataextractor.parse.actions.**flatten**(*tokens*, *start*, *result*)
Replace all child results with their text contents.

chemdataextractor.parse.actions.**join**(*tokens*, *start*, *result*)
Join tokens into a single string with spaces between.

chemdataextractor.parse.actions.**merge**(*tokens*, *start*, *result*)
Join tokens into a single string with no spaces.

chemdataextractor.parse.actions.**strip_stop**(*tokens*, *start*, *result*)
Remove trailing full stop from tokens.

chemdataextractor.parse.actions.**fix_whitespace**(*tokens*, *start*, *result*)
Fix whitespace around hyphens and commas. Can be used to remove whitespace tokenization artefacts.

3.8.2 .parse.auto

Parser for automatic parsing, without user-written parsing rules. Mainly used for tables.

Models must be constructed in a certain way for them to work optimally with autoparsers. Namely, they should have:

- A specifier field with an associated parse expression (Optional, only required if autoparsers are desired). These parse expressions will be updated automatically using forward-looking Interdependency Resolution if the updatable flag is set to True.
- These specifiers should also have required set to True so that spurious matches are not found.
- If applicable, a compound entity, named compound.

Any parse_expressions set in the model should have an added action to ensure that the results are a single word. An example would be to call add_action(join) on each parse expression.

chemdataextractor.parse.auto.**construct_unit_element**(*dimensions*)
Construct an element for detecting units for the dimensions given. Any magnitude modifiers (e.g. kilo) will be automatically handled.

Parameters **dimensions** (Dimension) – The dimensions that the element produced will look for.

Returns An Element to look for units of given dimensions. If None or Dimensionless are passed in, returns None.

Return type *BaseParserElement* or None

chemdataextractor.parse.auto.**construct_category_element**(*category_dict*)
Construct an element for detecting categories.

Parameters **category** (*Category*) – The Category to look for.

Return type *BaseParserElement* or None

chemdataextractor.parse.auto.**match_dimensions_of**(*model*)
Produces a function that checks whether the given results of parsing match the dimensions of the model provided.

Parameters **model** (QuantityModel) – The model with which to check dimensions.

Returns A function which will return True if the results of parsing match the model's dimensions, False if not.

Return type function(tuple(list(Element), int) -> bool)

chemdataextractor.parse.auto.**create_entities_list**(*entities*)
For a list of Base parser entities, creates an entity of structure. For example, with 4 entities in the list, the output is:

```
(entities[0] | entities[1] | entities[2] | entities[3])
```

Parameters **entities** – BaseParserElement type objects

Returns BaseParserElement type object

class chemdataextractor.parse.auto.**BaseAutoParser**
Bases: *chemdataextractor.parse.base.BaseParser*

model = None

__init__()
Initialize self. See help(type(self)) for accurate signature.

interpret(*result*, *start*, *end*)

class chemdataextractor.parse.auto.**AutoSentenceParser**(*lenient=False*, *chem_name=<chemdataextractor.parse.elements.First object>*)
Bases: *chemdataextractor.parse.auto.BaseAutoParser*, *chemdataextractor.parse.base.BaseSentenceParser*

__init__(*lenient=False*, *chem_name=<chemdataextractor.parse.elements.First object>*)
Initialize self. See help(type(self)) for accurate signature.

trigger_phrase

root

class chemdataextractor.parse.auto.**AutoTableParser**(*chem_name=<chemdataextractor.parse.elements.First object>*)
Bases: *chemdataextractor.parse.auto.BaseAutoParser*, *chemdataextractor.parse.base.BaseTableParser*

Additions for automated parsing of tables

__init__(*chem_name=<chemdataextractor.parse.elements.First object>*)
Initialize self. See help(type(self)) for accurate signature.

root

3.8.3 .parse.base

Base classes for parsing sentences and tables.

class chemdataextractor.parse.base.**BaseParser**

Bases: object

model = None

trigger_phrase = None

Optional *BaseParserElement* instance. All sentences are run through this before the full root phrase is applied to the sentence. If nothing is found for this phrase, the sentence will not go through the full root phrase. This is done for performance reasons, and if not set, ChemDataExtractor will perform as it did in previous versions. If this phrase is set to an appropriate value, it can help ChemDataExtractor perform at up to 2x its previous speed.

To ensure that this works as intended, the *BaseParserElement* should be a simple parse rule (substantially simpler than the *root*) that takes little time to process.

root

interpret(*result*, *start*, *end*)

extract_error(*string*)

Extract the error from a string

Usage:

```
bp = BaseParser()
test_string = '150±5'
end_value = bp.extract_error(test_string)
print(end_value) # 5
```

Parameters **string** (*str*) – A representation of the value and error as a string

Returns The error expressed as a float .

Return type float

extract_value(*string*)

Takes a string and returns a list of floats representing the string given.

Usage:

```
bp = BaseParser()
test_string = '150 to 160'
end_value = bp.extract_value(test_string)
print(end_value) # [150., 160.]
```

Parameters **string** (*str*) – A representation of the values as a string

Returns The value expressed as a list of floats of length 1 if the value had no range, and as a list of floats of length 2 if it was a range.

Return type list(float)

extract_units(*string*, *strict=False*)

Takes a string and returns a Unit. Raises TypeError if strict and the dimensions do not match the expected dimensions or the string has extraneous characters, e.g. if a string Fe was given, and we were looking for a temperature, strict=False would return Fahrenheit, strinct=True would raise a TypeError.

Usage:

```
bp = QuantityParser()
bp.model = QuantityModel()
bp.model.dimensions = Temperature() * Length()**0.5 * Time()**(1.5)
test_string = 'Kh2/(km/s)-1/2'
end_units = bp.extract_units(test_string, strict=True)
print(end_units) # Units of: (10^1.5) * Hour^(2.0)  Meter^(0.5)  Second^(-0.
↪5)  Kelvin^(1.0)
```

Parameters

- **string** (*str*) – A representation of the units as a string
- **strict** (*bool*) – Whether to raise a TypeError if the dimensions of the parsed units do not have the expected dimensions.

Returns The string expressed as a Unit

Return type chemdataextractor.quantities.Unit

class chemdataextractor.parse.base.**BaseSentenceParser**

Bases: *chemdataextractor.parse.base.BaseParser*

Base class for parsing sentences. To implement a parser for a new property, impelement the interpret function.

parse_sentence(*tokens*)

Parse a sentence. This function is primarily called by the *records* property of *Sentence*.

Parameters **tokens** (*list[(token,tag)]*) – List of tokens for parsing. When this method is called by *chemdataextractor.doc.text.Sentence.records*, the tokens passed in are *chemdataextractor.doc.text.Sentence.tagged_tokens*.

Returns All the models found in the sentence.

Return type Iterator[*chemdataextractor.model.base.BaseModel*]

class chemdataextractor.parse.base.**BaseTableParser**

Bases: *chemdataextractor.parse.base.BaseParser*

Base class for parsing new-style tables. To implement a parser for a new property, impelement the interpret function.

parse_cell(*cell*)

Parse a cell. This function is primarily called by the *records* property of *Table*.

Parameters **tokens** (*list[(token,tag)]*) – List of tokens for parsing. When this method is called by chemdataextractor.doc.text.table.Table, the tokens passed in are in the same form as *chemdataextractor.doc.text.Sentence.tagged_tokens*, after the category table has been flattened into a sentence.

Returns All the models found in the table.

Return type Iterator[*chemdataextractor.model.base.BaseModel*]

3.8.4 .parse.cem

Chemical entity mention parser elements. ..codeauthor:: Matt Swain (mcs07@cam.ac.uk) ..codeauthor:: Callum Court (cc889@cam.ac.uk)

chemdataextractor.parse.cem.**strict_chemical_label = <chemdataextractor.parse.elements.And**
Chemical label. Very permissive - must be used in context to avoid false positives.

chemdataextractor.parse.cem.**chemical_label_phrase1 = <chemdataextractor.parse.elements.And**
Chemical label with a label type before

chemdataextractor.parse.cem.**chemical_label_phrase2 = <chemdataextractor.parse.elements.And**
Chemical label with synthesis of before

chemdataextractor.parse.cem.**element_symbol = <chemdataextractor.parse.elements.Regex object**
Mostly unambiguous element symbols

chemdataextractor.parse.cem.**registry_number = <chemdataextractor.parse.elements.First objec**
Registry number patterns

chemdataextractor.parse.cem.**amino_acid = <chemdataextractor.parse.elements.Regex object>**
Amino acid abbreviations. His removed, too ambiguous

chemdataextractor.parse.cem.**formula = <chemdataextractor.parse.elements.First object>**
Chemical formula patterns, updated to include Inorganic compound formulae

chemdataextractor.parse.cem.**other_solvent = <chemdataextractor.parse.elements.First object>**
Solvent names.

chemdataextractor.parse.cem.**standardize_role**(*role*)
Convert role text into standardized form.

class chemdataextractor.parse.cem.**CompoundParser**
Bases: *chemdataextractor.parse.base.BaseSentenceParser*

Chemical name possibly with an associated label.

root

interpret(*result*, *start*, *end*)

class chemdataextractor.parse.cem.**ChemicalLabelParser**
Bases: *chemdataextractor.parse.base.BaseSentenceParser*

Chemical label occurrences with no associated name.

root

interpret(*result*, *start*, *end*)

class chemdataextractor.parse.cem.**CompoundHeadingParser**
Bases: *chemdataextractor.parse.base.BaseSentenceParser*

Better matching of abbreviated names in dedicated compound headings.

root = <chemdataextractor.parse.elements.Group object>

interpret(*result*, *start*, *end*)

class chemdataextractor.parse.cem.**CompoundTableParser**
Bases: *chemdataextractor.parse.base.BaseTableParser*

entities = <chemdataextractor.parse.elements.First object>

root

interpret(*result*, *start*, *end*)

3.8.5 .parse.common

Common parser elements.

3.8.6 .parse.context

3.8.7 .parse.elements

Parser elements.

exception chemdataextractor.parse.elements.**ParseException**(*tokens*, *i=0*, *msg=None*, *element=None*)

Bases: Exception

Exception thrown by a ParserElement when it doesn't match input.

__init__(*tokens*, *i=0*, *msg=None*, *element=None*)
Initialize self. See help(type(self)) for accurate signature.

classmethod wrap(*parse_exception*)

chemdataextractor.parse.elements.**safe_name**(*name*)
Make name safe for use in XML output.

class chemdataextractor.parse.elements.**BaseParserElement**
Bases: object

Abstract base parser element class.

__init__()
Initialize self. See help(type(self)) for accurate signature.

actions = None
name for BaseParserElement. This is used to set the name of the Element when a result is found

Type str or None

streamlined = None
list of actions that will be applied to the results after parsing. Actions are functions with arguments of (tokens, start, result)

Type list(chemdataextractor.parse.actions)

set_action(**fns*)

add_action(**fns*)

with_condition(*condition*)
Add a condition to the parser element. The condition must be a function that takes a match and return True or False, i.e. a function which takes tuple(list(Element), int) and returns bool. If the function evaluates True, the match is kept, while if the function evaluates False, the match is discarded. The condition is executed after any other actions.

copy()

set_name(*name*)

scan(*tokens*, *max_matches=9223372036854775807*, *overlap=False*)
Scans for matches in given tokens.

Parameters

- **string)) tokens** (*list(tuple(string,)*) – A tokenized representation of the text to scan. The first string in the tuple is the content, typically a word, and the second string is the part of speech tag.
- **max_matches** (*int*) – The maximum number of matches to look for. Default is the maximum size possible for a list.
- **overlap** (*bool*) – Whether the found results are allowed to overlap. Default False.

Returns A generator of the results found. Each result is a tuple with the first element being a list of elements found, and the second and third elements are the start and end indices representing the span of the result.

Return type generator(tuple(list(lxml.etree.Element), int, int))

parse(*tokens*, *i*, *actions=True*)

Parse given tokens and return results

Parameters

- **tokens** (*list(tuple(string, string))*) – A tokenized representation of the text to scan. The first string in the tuple is the content, typically a word, and the second string is the part of speech tag.
- **i** (*int*) – The index at which to start scanning from
- **actions** (*bool*) – Whether the actions attached to this element will be executed. Default True.

Returns A tuple where the first element is a list of elements found (can be None if no results were found), and the last index investigated.

Return type tuple(list(Element) or None, int)

try_parse(*tokens*, *i*)

streamline()

Streamlines internal representations. e.g., if we have something like And(And(And(And(a), b), c), d), streamline this to And(a, b, c, d)

hide()

class chemdataextractor.parse.elements.**Any**

Bases: *chemdataextractor.parse.elements.BaseParserElement*

Always match a single token.

class chemdataextractor.parse.elements.**NoMatch**

Bases: *chemdataextractor.parse.elements.BaseParserElement*

class chemdataextractor.parse.elements.**Word**(*match*)

Bases: *chemdataextractor.parse.elements.BaseParserElement*

Match token text exactly. Case-sensitive.

__init__(*match*)

Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**Tag**(*match*)

Bases: *chemdataextractor.parse.elements.BaseParserElement*

Match tag exactly.

__init__(*match*)

Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**IWord**(*match*)
Bases: *chemdataextractor.parse.elements.Word*

Case-insensitive match token text.

__init__(*match*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**Regex**(*pattern*, *flags=0*, *group=None*)
Bases: *chemdataextractor.parse.elements.BaseParserElement*

Match token text with regular expression.

__init__(*pattern*, *flags=0*, *group=None*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**Start**
Bases: *chemdataextractor.parse.elements.BaseParserElement*

Match at start of tokens.

__init__()
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**End**
Bases: *chemdataextractor.parse.elements.BaseParserElement*

Match at end of tokens.

__init__()
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**ParseExpression**(*exprs*)
Bases: *chemdataextractor.parse.elements.BaseParserElement*

Abstract class for combining and post-processing parsed tokens.

__init__(*exprs*)
Initialize self. See help(type(self)) for accurate signature.

append(*other*)

copy()

streamline()
Streamlines internal representations. e.g., if we have something like And(And(And(And(a), b), c), d), streamline this to And(a, b, c, d)

class chemdataextractor.parse.elements.**And**(*exprs*)
Bases: *chemdataextractor.parse.elements.ParseExpression*

Match all in the given order. Can probably be replaced by the plus operator '+'?

__init__(*exprs*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**Or**(*exprs*)
Bases: *chemdataextractor.parse.elements.ParseExpression*

Match the longest. Can probably be replaced by the pipe operator 'I'.

class chemdataextractor.parse.elements.**First**(*exprs*)
Bases: *chemdataextractor.parse.elements.ParseExpression*

Match the first.

__init__(*exprs*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**ParseElementEnhance**(*expr*)
Bases: *chemdataextractor.parse.elements.BaseParserElement*

Abstract class for combining and post-processing parsed tokens.

__init__(*expr*)
Initialize self. See help(type(self)) for accurate signature.

streamline()
Streamlines internal representations. e.g., if we have something like And(And(And(And(a), b), c), d), streamline this to And(a, b, c, d)

class chemdataextractor.parse.elements.**FollowedBy**(*expr*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

Check ahead if matches.

Example:

```
Tn + FollowedBy('Neel temperature')
Tn will match only if followed by 'Neel temperature', but 'Neel temperature' will␣
↪not be part of the output/tree
```

class chemdataextractor.parse.elements.**Not**(*expr*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

Check ahead to disallow a match with the given parse expression.

Example:

```
Tn + Not('some_string')
Tn will match if not followed by 'some_string'
```

class chemdataextractor.parse.elements.**ZeroOrMore**(*expr*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

Optional repetition of zero or more of the given expression.

class chemdataextractor.parse.elements.**OneOrMore**(*expr*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

Repetition of one or more of the given expression.

class chemdataextractor.parse.elements.**Optional**(*expr*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

Can be present but doesn't need to be. If present, will be added to the result/tree.

__init__(*expr*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**Group**(*expr*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

For nested tags; will group argument and give it a label, preserving the original sub-tags. Otherwise, the default behaviour would be to rename the outermost tag in the argument. Usage: Group(some_text)('new_tag) where 'some_text' is a previously tagged expression

class chemdataextractor.parse.elements.**SkipTo**(*expr*, *include=False*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

Skips to the next occurance of expression. Does not add the next occurance of expression to the parse tree. For example:

```
entities + SkipTo(entities)
```

will output `entities` only once. Whereas:

```
entities + SkipTo(entities) + entities
```

will output `entities` as well as the second occurrence of `entities` after an arbitrary number of tokens in between.

__init__(*expr*, *include=False*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.parse.elements.**Hide**(*expr*)
Bases: *chemdataextractor.parse.elements.ParseElementEnhance*

Converter for ignoring the results of a parsed expression. It wouldn't appear in the generated xml element tree, but it would still be part of the rule.

hide()

chemdataextractor.parse.elements.**W**
alias of *chemdataextractor.parse.elements.Word*

chemdataextractor.parse.elements.**I**
alias of *chemdataextractor.parse.elements.IWord*

chemdataextractor.parse.elements.**R**
alias of *chemdataextractor.parse.elements.Regex*

chemdataextractor.parse.elements.**T**
alias of *chemdataextractor.parse.elements.Tag*

chemdataextractor.parse.elements.**H**
alias of *chemdataextractor.parse.elements.Hide*

3.8.8 .parse.ir

IR spectrum text parser.

chemdataextractor.parse.ir.**extract_units**(*tokens*, *start*, *result*)
Extract units from bracketed after nu

class chemdataextractor.parse.ir.**IrParser**
Bases: *chemdataextractor.parse.base.BaseSentenceParser*

root = <chemdataextractor.parse.elements.And object>

interpret(*result*, *start*, *end*)

3.8.9 .parse.mp

NMR text parser.

class chemdataextractor.parse.mp.**MpParser**
Bases: *chemdataextractor.parse.base.BaseParser*

root = <chemdataextractor.parse.elements.First object>

interpret(*result*, *start*, *end*)

3.8.10 .parse.nmr

NMR text parser.

chemdataextractor.parse.nmr.**fix_nmr_peak_whitespace_error**(*tokens*, *start*, *result*)

chemdataextractor.parse.nmr.**strip_delta**(*tokens*, *start*, *result*)

class chemdataextractor.parse.nmr.**NmrParser**

Bases: *chemdataextractor.parse.base.BaseParser*

root = <chemdataextractor.parse.elements.And object>

__init__()

Initialize self. See help(type(self)) for accurate signature.

interpret(*result*, *start*, *end*)

3.8.11 .parse.template

Basic property parser template for Quantity Models

class chemdataextractor.parse.template.**QuantityModelTemplateParser**

Bases: *chemdataextractor.parse.auto.BaseAutoParser*, *chemdataextractor.parse.base.BaseSentenceParser*

Template parser for QuantityModel-type structures

Finds Cem, Specifier, Value and Units from single sentences

Other entities are merged contextually

specifier_phrase

The model specifier

value_phrase

Value and units

cem_phrase

CEM phrases

prefix

Specifier prefix phrase e.g. Tc equal to

specifier_and_value

Specifier and value + units

cem_before_specifier_and_value_phrase

Phrases ordered CEM, Specifier, Value, Unit

specifier_before_cem_and_value_phrase

cem_after_specifier_and_value_phrase

Phrases ordered specifier, value, unit, CEM

value_specifier_cem_phrase

Phrases ordered value unit specifier cem

root

Root Phrases

class chemdataextractor.parse.template.**MultiQuantityModelTemplateParser**

Bases: *chemdataextractor.parse.auto.BaseAutoParser*, *chemdataextractor.parse.base.BaseSentenceParser*

Template for parsing sentences that contain nested or chained entities

MULTIPLE ENTITY PHRASES

1) Single compound, multiple specifiers, multiple phase transitions e.g. BiFeO3 has TC = 1093 K and TN = 640 K
2) single compound, single specifier, multiple transitions e.g. BiFeO3 shows magnetic transitions at 1093 and 640 K
3) multiple compounds, single specifier, multiple transitions e.g. TC in BiFeO3 and LaFeO3 of 640 and 750 K
4) multiple compounds, single specifier, single transition e.g. TC of 640 K in BifEO3, LaFeO3 and MnO
5) multiple compounds, multiple specifiers, multiple transitions e.g. BiFeO3 and LaFeO3 have Tc = 640 K and TN = 750 K respectively

Parameters

- **{[type]} -- [description]** (BaseSentenceParser) –
- **{[type]} -- [description]** –

specifier_phrase
Specifier Phrase

prefix
Specifier and prefix

single_cem
Any cem

unit
Unit element

value_with_optional_unit
Value possibly followed by a unit

value_phrase
Value with unit

list_of_values
List of values with either multiple units or one at the end

list_of_cems
List of cems e.g. cem1, cem2, cem3 and cem4

single_specifier_and_value_with_optional_unit
Specifier plus value and possible unit

single_specifier_and_value
Specifier value and unit

list_of_properties
List of specifiers and units

multi_entity_phrase_1
Single compound, multiple specifiers, values e.g. BiFeO3 has TC1 = 1093 K and Tc2 = 640 K

multi_entity_phrase_2
single compound, single specifier, multiple transitions e.g. BiFeO3 shows magnetic transitions at 1093 and 640 K

multi_entity_phrase_3a
multiple compounds, single specifier, multiple transitions cems first e.g. TC in BiFeO3 and LaFeO3 of 640 and 750 K

multi_entity_phrase_3b
multiple compounds, single specifier, multiple transitions cems last e.g. Tc = 750 and 640 K in LaFeO3 and BiFeO3, respectivel

multi_entity_phrase_3c
multiple compounds, single specifier, multiple transitions cems last e.g. curie temperatures from 100 K in MnO to 300 K in NiO

multi_entity_phrase_3
Combined phrases of type 3

multi_entity_phrase_4a
multiple compounds, single specifier, single transition e.g. TC of 640 K in BifEO3, LaFeO3 and MnO

multi_entity_phrase_4b
Cems first

multi_entity_phrase_4

root

interpret(*result*, *start*, *end*)

interpret_multi_entity_1(*result*, *start*, *end*)
Interpret phrases that have a single CEM and multiple values with multiple specifiers

interpret_multi_entity_2(*result*, *start*, *end*)
single compound, single specifier, multiple transitions e.g. BiFeO3 shows magnetic transitions at 1093 and 640 K

interpret_multi_entity_3(*result*, *start*, *end*)
interpret multiple compounds, single specifier, multiple transitions

interpret_multi_entity_4(*result*, *start*, *end*)
interpret multiple compounds, single specifier, single transition

3.8.12 .parse.tg

Glass transition temperature parser.

class chemdataextractor.parse.tg.**TgParser**
Bases: *chemdataextractor.parse.base.BaseParser*

root = <chemdataextractor.parse.elements.First object>

interpret(*result*, *start*, *end*)

3.8.13 .parse.uvvis

UV-vis text parser.

class chemdataextractor.parse.uvvis.**UvvisParser**
Bases: *chemdataextractor.parse.base.BaseSentenceParser*

root = <chemdataextractor.parse.elements.And object>

interpret(*result*, *start*, *end*)

3.9 .reader

Document readers

Reader classes that read a file and produce a ChemDataExtractor Document object.

3.9.1 .reader.acs

Readers for documents from the ACS.

chemdataextractor.reader.acs.**clean_acs_html = <chemdataextractor.scrape.clean.Cleaner objec**
Move to ignore_css?

Type Additional cleaner for ACS HTML TODO

class chemdataextractor.reader.acs.**AcsHtmlReader**
Bases: *chemdataextractor.reader.markup.HtmlReader*

Reader for HTML documents from the ACS.

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>, <chemdataextractor.scrape

root_css = '#articleMain, article'

title_css = 'h1.articleTitle'

heading_css = 'h2, h3, h4, h5, h6, .title1, span.title2, span.title3'

table_css = '.NLM_table-wrap'

table_caption_css = '.NLM_caption'

table_footnote_css = '.footnote'

figure_css = '.figure'

figure_caption_css = '.caption'

citation_css = '.reference'

ignore_css = 'a[href="JavaScript:void(0);"], a.ref sup'

detect(*fstring*, *fname=None*)

3.9.2 .reader.base

Abstract base classes for document readers.

class chemdataextractor.reader.base.**BaseReader**
Bases: object

All Document Readers should implement a parse method.

__init__()
Initialize self. See help(type(self)) for accurate signature.

detect (*fstring*, *fname=None*)
Quickly check if this reader can parse the input. Reader subclasses should override this.

Used to quickly skip attempting to parse when trying different readers. If in doubt, return True, and then raise ReaderError in the parse method if it fails.

parse (*fstring*)
Parse the input and return a Document. Raises ReaderError if the parse fails.

read (*f*)
Read a file-like object and return a Document.

readstring (*fstring*)
Read a file string and return a Document.

3.9.3 .reader.cssp

Readers for ChemSpider SyntheticPages.

class chemdataextractor.reader.cssp.**CsspHtmlReader**
Bases: *chemdataextractor.reader.markup.HtmlReader*

Reader for ChemSpider SyntheticPages HTML documents.

root_css = '.article-container'

title_css = '.article-container > h2'

heading_css = 'h3, h4, h5, h6'

citation_css = '#csm-article-part-lead_ref > p, #csm-article-part-other_refs > p'

detect (*fstring*, *fname=None*)

3.9.4 .reader.markup

XML and HTML readers based on lxml.

class chemdataextractor.reader.markup.**LxmlReader**
Bases: *chemdataextractor.reader.base.BaseReader*

Abstract base class for lxml-based readers.

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>]
A Cleaner instance to

root_css = 'html'

title_css = 'h1'

heading_css = 'h2, h3, h4, h5, h6'

table_css = 'table'

table_caption_css = 'caption'

table_head_row_css = 'thead tr'

table_body_row_css = 'tbody tr'

table_cell_css = 'th, td'

table_footnote_css = 'tfoot tr th'

reference_css = 'a.ref'

figure_css = 'figure'

figure_caption_css = 'figcaption'

figure_label_css = 'figcaption span[class^="CaptionNumber"]'

figure_download_link_css = 'a::attr(href), img::attr(src)'

citation_css = 'cite'

metadata_css = 'head'

metadata_publisher_css = 'meta[name="DC.publisher"]::attr("content"), meta[name="citat

metadata_author_css = 'meta[name="DC.Creator"]::attr("content"), meta[name="citation_a

metadata_title_css = 'meta[name="DC.title"]::attr("content"), meta[name="citation_titl

metadata_date_css = 'meta[name="DC.Date"]::attr("content"), meta[name="citation_date"]

metadata_doi_css = 'meta[name="DC.Identifier"]::attr("content"), meta[name="citation_d

metadata_language_css = 'meta[name="DC.Language"]::attr("content"), meta[name="citatio

metadata_journal_css = 'meta[name="citation_journal_title"]::attr("content")'

metadata_volume_css = 'meta[name="citation_volume"]::attr("content")'

metadata_issue_css = 'meta[name="citation_issue"]::attr("content")'

metadata_firstpage_css = 'meta[name="citation_firstpage"]::attr("content")'

metadata_lastpage_css = 'meta[name="citation_lastpage"]::attr("content")'

metadata_pdf_url_css = 'meta[name="citation_pdf_url"]::attr("content")'

metadata_html_url_css = 'meta[name="citation_fulltext_html_url"]::attr("content"), meta

ignore_css = 'a.ref sup'

inline_elements = {'a', 'abbr', 'acronym', 'b', 'bdo', 'big', 'blink', 'br', 'button',
Inline elements

parse(*fstring*)
Parse the input and return a Document. Raises ReaderError if the parse fails.

class chemdataextractor.reader.markup.**XmlReader**
Bases: *chemdataextractor.reader.markup.LxmlReader*

Reader for generic XML documents.

detect(*fstring*, *fname=None*)

class chemdataextractor.reader.markup.**HtmlReader**
Bases: *chemdataextractor.reader.markup.LxmlReader*

Reader for generic HTML documents.

detect(*fstring*, *fname=None*)

3.9.5 .reader.nlm

Readers for NLM Journal Archiving and Interchange DTD XML files. (i.e. from PubMed Central)

class chemdataextractor.reader.nlm.**NlmXmlReader**
Bases: *chemdataextractor.reader.markup.XmlReader*

Reader for NLM XML documents.

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>, <function tidy_nlm_refere

root_css = 'article'

title_css = 'front article-meta article-title'

heading_css = 'title'

table_css = 'table-wrap'

table_caption_css = 'caption p'

table_head_row_css = 'table thead tr'

table_body_row_css = 'table tbody tr'

table_footnote_css = 'table-wrap-foot p'

figure_css = 'fig'

figure_caption_css = 'caption p'

reference_css = 'xref'

citation_css = 'ref-list ref'

ignore_css = 'xref[ref-type="bibr"], tex-math'

inline_elements = {'a', 'abbr', 'acronym', 'alternatives', 'b', 'bdo', 'big', 'blink',

detect(*fstring*, *fname=None*)

3.9.6 .reader.pdf

PDF document reader.

class chemdataextractor.reader.pdf.**PdfReader**
Bases: *chemdataextractor.reader.base.BaseReader*

detect(*fstring*, *fname=None*)

parse(*fstring*)
Parse the input and return a Document. Raises ReaderError if the parse fails.

3.9.7 .reader.plaintext

Plain text document reader.

class chemdataextractor.reader.plaintext.**PlainTextReader**
Bases: *chemdataextractor.reader.base.BaseReader*

Read plain text and split into Paragraphs based on newline patterns.

detect(*fstring*, *fname=None*)
Have a stab at most files.

parse(*fstring*)
Parse the input and return a Document. Raises ReaderError if the parse fails.

3.9.8 .reader.rsc

Readers for documents from the RSC.

chemdataextractor.reader.rsc.**rsc_html_whitespace**(*document*)
Remove whitespace in xml.text or xml.tails for all elements, if it is only whitespace

chemdataextractor.reader.rsc.**join_rsc_table_captions**(*document*)
Add wrapper tag around Tables and their respective captions

Parameters **{[type]} -- [description]** (*document*) –

class chemdataextractor.reader.rsc.**RscHtmlReader**
Bases: *chemdataextractor.reader.markup.HtmlReader*

Reader for HTML documents from the RSC.

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>, <function rsc_html_whites

root_css = 'html'

title_css = 'h1, .title_heading'

heading_css = 'h2, h3, h4, h5, h6, .a_heading, .b_heading, .c_heading, .c_heading_inde

citation_css = 'span[id^="cit"]'

table_css = 'div[class^="rtable__wrapper"]'

table_caption_css = '.table_caption'

table_head_row_css = 'table thead tr'

table_body_row_css = 'table tbody tr'

table_footnote_css = 'table tfoot tr th .sup_inf'

reference_css = 'small sup a, a[href^="#cit"], a[href^="#fn"], a[href^="#tab"]'

figure_css = '.image_table'

figure_caption_css = '.graphic_title'

figure_label_css = 'td.image_title b'

figure_download_link_css = 'img::attr(src)'

ignore_css = '.table_caption + table, .left_head, sup span.sup_ref, small sup a, a[hre

detect(*fstring*, *fname=None*)

3.9.9 .reader.uspto

Readers for USPTO patents.

class chemdataextractor.reader.uspto.**UsptoXmlReader**
Bases: *chemdataextractor.reader.markup.XmlReader*

Reader for USPTO XML documents.

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>]

root_css = 'us-patent-grant'

title_css = 'invention-title'

heading_css = 'heading, p[id^="h-"]'

table_css = 'table'

table_body_row_css = 'table row'

table_cell_css = 'entry'

reference_css = 'claim-ref'

ignore_css = 'us-bibliographic-data-grant *:not(invention-title)'

inline_elements = {'a', 'abbr', 'acronym', 'alternatives', 'b', 'bdo', 'big', 'blink',

detect(*fstring*, *fname=None*)

3.9.10 .reader.elsevier

Elsevier XML reader

Readers for Elsevier XML files.

chemdataextractor.reader.elsevier.**fix_elsevier_xml_whitespace**(*document*)
Fix tricky xml tags

chemdataextractor.reader.elsevier.**els_xml_whitespace**(*document*)
Remove whitespace in xml.text or xml.tails for all elements, if it is only whitespace

class chemdataextractor.reader.elsevier.**ElsevierXmlReader**
Bases: *chemdataextractor.reader.markup.XmlReader*

Reader for Elsevier XML documents.

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>, <function fix_elsevier_xm

root_css = 'default|full-text-retrieval-response'

title_css = 'dc|title'

heading_css = 'ce|section-title'

table_css = 'ce|table'

table_caption_css = 'ce|table ce|caption'

table_head_row_css = 'cals|thead cals|row'

table_body_row_css = 'cals|tbody cals|row'

table_cell_css = 'ce|entry'

table_footnote_css = 'table-wrap-foot p'

figure_css = 'ce|figure'

figure_caption_css = 'ce|figure ce|caption'

figure_label_css = 'ce|figure ce|label'

figure_download_link_css = ''

reference_css = 'ce|cross-ref, ce|cross-refs'

citation_css = 'ce|bib-reference'

metadata_css = 'xocs|meta'

metadata_title_css = 'xocs|normalized-article-title'

metadata_author_css = 'xocs|normalized-first-auth-surname'

metadata_journal_css = 'xocs|srctitle'

metadata_volume_css = 'xocs|vol-first, xocs|volume-list xocs|volume'

metadata_issue_css = 'xocs|issns xocs|issn-primary-formatted'

metadata_publisher_css = 'xocs|copyright-line'

metadata_date_css = 'xocs|available-online-date, xocs|orig-load-date'

metadata_firstpage_css = 'xocs|first-fp'

metadata_lastpage_css = 'xocs|last-lp'

metadata_doi_css = 'xocs|doi, xocs|eii'

metadata_pii_css = 'xocs|pii-unformatted'

ignore_css = 'ce|bibliography, ce|acknowledgment, ce|correspondence, ce|author, ce|doi

url_prefix = 'https://sciencedirect.com/science/article/pii/'

detect(*fstring*, *fname=None*)
Elsevier document detection based on string found in xml

3.9.11 .reader.springer

Readers for documents from Springer.

class chemdataextractor.reader.springer.**SpringerMaterialsHtmlReader**
Bases: *chemdataextractor.reader.markup.HtmlReader*

Reader for HTML documents from SpringerMaterials.

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>, <chemdataextractor.scrape

root_css = 'html'

citation_css = 'span[class="CitationRef"]'

title_css = 'title'

heading_css = 'h2, h3, h4, h5, h6, .title1, span.title2, span.title3'

table_css = 'div[class="Table"]'

table_caption_css = 'div[class="Table"] p'

table_head_row_css = 'thead'

table_body_row_css = 'tbody'

table_cell_css = 'th, td'

ignore_css = 'sub, sup, em[class^="EmphasisTypeItalic "], li[class="article-metrics__i

detect(*fstring*, *fname=None*)

chemdataextractor.reader.springer.**springer_html_whitespace**(*document*)
Remove whitespace in xml.text or xml.tails for all elements, if it is only whitespace

chemdataextractor.reader.springer.**fix_springer_table_whitespace**(*document*)
remove leading and trailing whitespace from table cells

Parameters **{[type]} -- [description]**(*document*) –

Returns [type] – [description]

class chemdataextractor.reader.springer.**SpringerHtmlReader**
Bases: *chemdataextractor.reader.markup.HtmlReader*

cleaners = [<chemdataextractor.scrape.clean.Cleaner object>, <function springer_html_w

root_css = 'html'

title_css = 'h1[class^="ArticleTitle"]'

heading_css = 'h2, h3, h4'

table_css = 'div[class="Table"]'

table_caption_css = 'div[class^="Caption"] p'

table_head_row_css = 'thead tr'

table_body_row_css = 'tbody tr'

table_cell_css = 'td, th'

figure_css = 'figure'

figure_caption_css = 'figcaption'

figure_label_css = 'figcaption span[class^="CaptionNumber"]'

ignore_css = 'a[class="skip-to__link pseudo-focus"], div[class="nojs-banner u-interfac

detect(*fstring*, *fname=None*)

3.10 .relex

For performing semi-supervised chemical Relationship Extraction using the Snowball Algorithm.

3.10.1 .relex.cluster

Cluster of phrase objects and associated cluster dictionaries

class chemdataextractor.relex.cluster.**Cluster**(*label=None*, *learning_rate=0.5*)
Bases: object

Base Snowball Cluster, used to combine similar phrases

__init__(*label=None*, *learning_rate=0.5*)
Create a new cluster

Keyword Arguments

- **{str} -- The label of this cluster (default** (*label*) – {None})
- **{list} -- The order of entities that all phrases in this cluster must share (default** (*order*) – {None})
- **{float} -- How quickly to update confidences based on new information (default** (*learning_rate*) – {0.5})

add_phrase(*phrase*)
Add phrase to this cluster, update the word dictionary and token weights

Parameters **phrase** (chemdataextractor.relex.phrase.Phrase) – The phrase to add to the cluster

update_dictionaries(*phrase*)
Update all dictionaries in this cluster

Parameters **phrase** (chemdataextractor.relex.phrase.Phrase) – The phrase to update

static add_tokens(*dictionary*, *tokens*)
Add specified tokens to the specified dictionary

Parameters

- **dictionary** (*OrderedDict*) – The dictionary to add tokens to
- **tokens** – tokens to add

Type list of str

update_weights()
Update the weights on each token in the phrases

update_pattern()
Use the cluster phrases to generate a new centroid extraction Pattern object

Parameters

- **relations** – List of known relations to look for
- **sentences** (*List of str*) – List of sentences known to contain relations

Type list of Relation objects

update_pattern_confidence()
Determine the confidence of this centroid pattern

get_relations(*tokens*)
Retrieve relations from a set of tokens using this clusters extraction pattern

Parameters **{list} -- Tokens to extract from**(*tokens*) –

Returns Relations – The found Relations

3.10.2 .relex.entity

Extraction pattern object

class chemdataextractor.relex.entity.**Entity**(*text*, *tag*, *parse_expression*, *start*, *end*)
Bases: object

A base entity, the fundamental unit of a Relation

__init__(*text*, *tag*, *parse_expression*, *start*, *end*)
Create a new Entity

Parameters

- **{str} -- The text of the entity**(*text*) –
- **{str or list} -- name of the entity**(*tag*) –
- **-- how the entity is identified in text**(*parse_expression*) –
- **{int} -- The index of the Entity in tokens**(*start*) –
- **{int} -- The end index of the entity in tokens**(*end*) –

serialize()

3.10.3 .relex.pattern

Extraction pattern object

class chemdataextractor.relex.pattern.**Pattern**(*entities=None*, *elements=None*, *label=None*, *sentences=None*, *order=None*, *relations=None*, *confidence=0*)

Bases: object

Pattern object, fundamentally the same as a phrase except assigned a confidence

__init__(*entities=None*, *elements=None*, *label=None*, *sentences=None*, *order=None*, *relations=None*, *confidence=0*)
Initialize self. See help(type(self)) for accurate signature.

to_string()

generate_cde_parse_expression()
Create a CDE parse expression for this extraction pattern

3.10.4 .relex.phrase

Phrase object

class chemdataextractor.relex.phrase.**Phrase**(*sentence_tokens*, *relations*, *prefix_length*, *suffix_length*)

Bases: object

__init__(*sentence_tokens*, *relations*, *prefix_length*, *suffix_length*)
Phrase Object

Class for handling which relations and entities appear in a sentence, the base type used for clustering and generating extraction patterns

Parameters

- **{[list} -- The sentence tokens from which to generate the Phrase** (*sentence_tokens*) –
- **{list} -- List of Relation objects to be tagged in the sentence** (*relations*) –
- **{int} -- Number of tokens to assign to the prefix** (*prefix_length*) –
- **{int} -- Number of tokens to assign to the suffix** (*suffix_length*) –

to_string()

create()
Create a phrase from known relations

reset_vectors()
Set all element vectors to None

3.10.5 .relex.relationship

Classes for defining new chemical relationships

class chemdataextractor.relex.relationship.**Relation**(*entities*, *confidence*)
Bases: object

Relation class

Essentially a placeholder for related of entities

__init__(*entities*, *confidence*)
Init

Parameters

- **{list} -- List of Entity objects that are present in this relationship**(*entities*) –
- **{float} -- The confidence of the relation**(*confidence*) –

serialize()

is_valid()

3.10.6 .relex.snowball

3.10.7 .relex.utils

Various utility functions

chemdataextractor.relex.utils.**match_score**(*pi*, *pj*, *prefix_weight=0.1*, *middle_weight=0.8*, *suffix_weight=0.1*)
Compute match between phrases using a dot product of vectors :param pi Phrase or pattern :param pj phrase or pattern # add weights to dot products to put more emphasis on matching the middles

chemdataextractor.relex.utils.**vectorise**(*phrase*, *cluster*)
Vectorise a phrase object against a given cluster

Parameters

- **{[type]} -- [description]**(*cluster*) –
- **{[type]} -- [description]** –

chemdataextractor.relex.utils.**match**(*phrase*, *cluster*, *prefix_weight*, *middles_weight*, *suffix_weight*)
Vectorise the phrase against this cluster to determine the match score

Parameters

- **{[type]} -- [description]**(*cluster*) –
- **{[type]} -- [description]** –

chemdataextractor.relex.utils.**mode_rows**(*a*)
Find the modal row of a 2d array :param a: The 2d array to process :type a: np.array() :return: The most frequent row

chemdataextractor.relex.utils.**KnuthMorrisPratt**(*text*, *pattern*)

Yields all starting positions of copies of the pattern in the text. Calling conventions are similar to string.find, but its arguments can be lists or iterators, not just strings, it returns all matches, not just the first one, and it does not need the whole text in memory at once. Whenever it yields, it will have read the text exactly up to and including the match that caused the yield.

Source: http://code.activestate.com/recipes/117214/

chemdataextractor.relex.utils.**subfinder**(*mylist*, *pattern*)

3.11 .scrape

Scrapers for the various data sources

Declarative scraping framework for extracting structured data from HTML and XML documents.

chemdataextractor.scrape.**BLOCK_ELEMENTS = {'address', 'article', 'aside', 'audio', 'blockqu**
Block level HTML elements

chemdataextractor.scrape.**INLINE_ELEMENTS = {'a', 'abbr', 'acronym', 'b', 'bdo', 'big', 'bli**
Inline level HTML elements

3.11.1 .scrape.pub

Scraping tools for specific publishers.

.scrape.pub.nlm

Tools for scraping documents from NLM Journal Archiving and Interchange DTD XML files.

chemdataextractor.scrape.pub.nlm.**strip_pmc_xml = <chemdataextractor.scrape.clean.Cleaner ob**
XML stripper that kills reference links, footnote links, equations, footnotes

chemdataextractor.scrape.pub.nlm.**strip_pmc_abstract_xml = <chemdataextractor.scrape.clean.C**
XML stripper that also kills headings

chemdataextractor.scrape.pub.nlm.**strip_pmc_paragraph_xml = <chemdataextractor.scrape.clean**
XML stripper that also kills tables and figures

chemdataextractor.scrape.pub.nlm.**space_labels**(*document*)
Ensure space around bold compound labels.

chemdataextractor.scrape.pub.nlm.**tidy_nlm_references**(*document*)
Remove punctuation around references like brackets, commas, hyphens.

class chemdataextractor.scrape.pub.nlm.**NlmXmlAuthor**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Author information from NLM XML file.

givennames
A string field.

lastname
A string field.

email
A string field.

process_givennames = <chemdataextractor.text.normalize.Normalizer object>

process_lastname = <chemdataextractor.text.normalize.Normalizer object>

fields = {'email': <chemdataextractor.scrape.fields.StringField object>, 'givennames'

class chemdataextractor.scrape.pub.nlm.**NlmXmlImage**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Figure information from NLM XML file.

label
A string field.

caption
A string field.

reference
A string field.

clean_caption = <chemdataextractor.text.processors.Chain object>

process_caption = <chemdataextractor.text.normalize.Normalizer object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'label':

class chemdataextractor.scrape.pub.nlm.**NlmXmlTable**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Table information from NLM XML file.

label
A string field.

caption
A string field.

reference
A string field.

src
A string field.

clean_caption = <chemdataextractor.text.processors.Chain object>

process_caption = <chemdataextractor.text.normalize.Normalizer object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'label':

class chemdataextractor.scrape.pub.nlm.**NlmXmlDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Document information from a NLM XML file.

doi
A string field.

pmid
An integer number field.

pmcid
An integer number field.

title
A string field.

authors
A field that contains another Entity.

journal_title
A string field.

journal_abbreviation
A string field.

publisher
A string field.

volume
A string field.

firstpage
A string field.

lastpage
A string field.

issue
A string field.

issn
A string field.

coden
A string field.

abstract
A string field.

online_year
An integer number field.

online_month
An integer number field.

online_day
An integer number field.

published_year
An integer number field.

published_month
An integer number field.

published_day
An integer number field.

accepted_year
An integer number field.

accepted_month
An integer number field.

accepted_day
An integer number field.

received_year
An integer number field.

received_month
An integer number field.

received_day
An integer number field.

license
A field with optional URL processing.

clean_title = <chemdataextractor.scrape.clean.Cleaner object>

clean_abstract = <chemdataextractor.scrape.clean.Cleaner object>

process_title = <chemdataextractor.text.normalize.Normalizer object>

process_publisher = <chemdataextractor.text.normalize.Normalizer object>

process_abstract = <chemdataextractor.text.normalize.Normalizer object>

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'accepted

.scrape.pub.rsc

Tools for scraping documents from The Royal Society of Chemistry.

chemdataextractor.scrape.pub.rsc.**CHAR_REPLACEMENTS = [('\\[?\\[1 with combining macron\\]**
Map placeholder text to unicode characters.

chemdataextractor.scrape.pub.rsc.**RSC_IMG_CHARS = {'2041': '^', '224a': '', 'e001': '=',**
Map image URL components to unicode characters.

chemdataextractor.scrape.pub.rsc.**strip_rsc_html = <chemdataextractor.scrape.clean.Cleaner o**
none;" (typically tooltips)

Type HTML stripper that kills superscript references and anything with style="display

chemdataextractor.scrape.pub.rsc.**strip_cit_html = <chemdataextractor.scrape.clean.Cleaner o**
HTML stripper that also kills text from buttons in references.

chemdataextractor.scrape.pub.rsc.**rsc_substitute = <chemdataextractor.text.processors.Substi**
Substitutor that replaces RSC escape codes with the actual unicode character

chemdataextractor.scrape.pub.rsc.**parse_rsc_html**(*htmlstring*)
Messy RSC HTML needs this special parser to fix problems before creating selector.

chemdataextractor.scrape.pub.rsc.**replace_rsc_img_chars**(*document*)
Replace image characters with unicode equivalents.

chemdataextractor.scrape.pub.rsc.**space_references**(*document*)
Ensure a space around reference links, so there's a gap when they are removed.

class chemdataextractor.scrape.pub.rsc.**RscRssDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Document information from RSC RSS feed.

doi
A string field.

title
A string field.

authors
A string field.

landing_url
A field with optional URL processing.

process_title = <chemdataextractor.text.processors.Chain object>

finalize_doi(*value*)
Derive the DOI from the GUID.

fields = {'authors': <chemdataextractor.scrape.fields.StringField object>, 'doi': <c

class chemdataextractor.scrape.pub.rsc.**RscRssScraper**
Bases: *chemdataextractor.scrape.scraper.RssScraper*

Scraper for RSC RSS feeds.

entity
alias of *RscRssDocument*

class chemdataextractor.scrape.pub.rsc.**RscSearchDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Document information from RSC search results page.

doi
A string field.

title
A string field.

landing_url
A field with optional URL processing.

pdf_url
A field with optional URL processing.

html_url
A field with optional URL processing.

journal
A string field.

abstract
A string field.

clean_title = <chemdataextractor.text.processors.Chain object>

process_doi = <chemdataextractor.text.processors.LAdd object>

process_title = <chemdataextractor.text.processors.Chain object>

process_landing_url = <chemdataextractor.text.processors.Chain object>

process_pdf_url = <chemdataextractor.text.processors.Chain object>

process_html_url = <chemdataextractor.text.processors.Chain object>

process_abstract = <chemdataextractor.text.processors.Chain object>

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'doi': <

class chemdataextractor.scrape.pub.rsc.**RscSearchScraper**(*max_wait_time=30*, *driver=None*)
Bases: *chemdataextractor.scrape.scraper.SearchScraper*

Scraper for RSC search results.

entity
alias of *RscSearchDocument*

root = '.capsule.capsule--article'

__init__(*max_wait_time=30*, *driver=None*)

Parameters

- **driver** (*selenium.webdriver*) – driver from which results will be scraped.
- **max_wait_time** (*float*) – Maximum time spent waiting for the page to load. (seconds)

perform_search(*query*, *page=1*, *driver=None*)
Due to RSC not accepting html requests, Selenium is used. By default, the Firefox webdriver is used.

class chemdataextractor.scrape.pub.rsc.**RscLandingSupplement**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

name
A string field.

url
A field with optional URL processing.

fields = {'name': <chemdataextractor.scrape.fields.StringField object>, 'url': <chem

class chemdataextractor.scrape.pub.rsc.**RscLandingDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.DocumentEntity*

Document information from RSC landing page.

supplements
A field that contains another Entity.

process_abstract = <chemdataextractor.text.processors.Chain object>

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'authors'

class chemdataextractor.scrape.pub.rsc.**RscLandingScraper**
Bases: *chemdataextractor.scrape.scraper.UrlScraper*

Scraper for RSC Landing pages.

entity
alias of *RscLandingDocument*

class chemdataextractor.scrape.pub.rsc.**RscChemicalMention**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

text
A string field.

chemspider_id
A string field.

inchi
A string field.

clean_text = <chemdataextractor.text.processors.Chain object>

process_text = <chemdataextractor.text.normalize.Normalizer object>

process_chemspider_id = <chemdataextractor.text.processors.Chain object>

process_inchi = <chemdataextractor.text.processors.Chain object>

fields = {'chemspider_id': <chemdataextractor.scrape.fields.StringField object>, 'inc

class chemdataextractor.scrape.pub.rsc.**RscImage**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Embedded image. Includes both Schemes and Figures.

url
A field with optional URL processing.

label
A string field.

reference
A string field.

caption
A string field.

clean_caption = <chemdataextractor.text.processors.Chain object>

process_caption = <chemdataextractor.text.normalize.Normalizer object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'label':

class chemdataextractor.scrape.pub.rsc.**RscTable**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Table within document.

reference
A string field.

label
A string field.

caption
A string field.

src
A string field.

clean_src = <chemdataextractor.text.processors.Chain object>

clean_caption = <chemdataextractor.text.processors.Chain object>

process_caption = <chemdataextractor.text.normalize.Normalizer object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'label':

class chemdataextractor.scrape.pub.rsc.**RscHtmlDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.DocumentEntity*

title
A string field.

abstract
A string field.

pdf_url
A field with optional URL processing.

html_url
A field with optional URL processing.

landing_url
A field with optional URL processing.

clean_title = <chemdataextractor.text.processors.Chain object>

clean_abstract = <chemdataextractor.text.processors.Chain object>

process_title = <chemdataextractor.text.processors.Chain object>

process_abstract = <chemdataextractor.text.normalize.Normalizer object>

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'authors'

class chemdataextractor.scrape.pub.rsc.**RscHtmlScraper**
Bases: *chemdataextractor.scrape.scraper.UrlScraper*

Scraper for RSC Landing pages.

entity
alias of *RscHtmlDocument*

.scrape.pub.springer

Tools for scraping documents from Springer, Biomed Central and Chemistry Central XML files.

chemdataextractor.scrape.pub.springer.**strip_springer_xml = <chemdataextractor.scrape.clean**
XML stripper that also kills equations/formulas.

chemdataextractor.scrape.pub.springer.**strip_springer_abstract_xml = <chemdataextractor.scra**
XML stripper that also kills headings

chemdataextractor.scrape.pub.springer.**tidy_springer_references**(*document*)
Remove punctuation around references like brackets, commas, hyphens.

class chemdataextractor.scrape.pub.springer.**SpringerHtmlDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.DocumentEntity*

Scraper for Springer HTML articles

title
A string field.

abstract
A string field.

journal
A string field.

process_html_url = <chemdataextractor.text.processors.RAdd object>

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'authors'

class chemdataextractor.scrape.pub.springer.**SpringerXmlAuthor**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Author information from a Springer XML file.

firstname
A string field.

middlename
A string field.

lastname
A string field.

suffix
A string field.

email
A string field.

process_email = <chemdataextractor.text.processors.Discard object>

fields = {'email': <chemdataextractor.scrape.fields.StringField object>, 'firstname':

class chemdataextractor.scrape.pub.springer.**SpringerXmlImage**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Figure information from a Springer XML file.

label
A string field.

caption
A string field.

reference
A string field.

clean_caption = <chemdataextractor.scrape.clean.Cleaner object>

process_caption = <chemdataextractor.text.normalize.Normalizer object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'label':

class chemdataextractor.scrape.pub.springer.**SpringerXmlTable**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Table information from a Springer XML file.

label
A string field.

caption
A string field.

reference
A string field.

src
A string field.

clean_caption = <chemdataextractor.scrape.clean.Cleaner object>

process_caption = <chemdataextractor.text.normalize.Normalizer object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'label':

class chemdataextractor.scrape.pub.springer.**SpringerXmlDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Document information from a Springer XML file.

ui
A string field.

doi
A string field.

title
A string field.

authors
A field that contains another Entity.

journal
A string field.

firstpage
A string field.

year
An integer number field.

volume
A string field.

issue
A string field.

issn
A string field.

landing_url
A field with optional URL processing.

abstract
A string field.

published_year
An integer number field.

published_month
An integer number field.

published_day
An integer number field.

accepted_year
An integer number field.

accepted_month
An integer number field.

accepted_day
An integer number field.

received_year
An integer number field.

received_month
An integer number field.

received_day
An integer number field.

license
A field with optional URL processing.

figures
A field that contains another Entity.

schemes
A field that contains another Entity.

tables
A field that contains another Entity.

headings
A string field.

paragraphs
A string field.

clean_title = <chemdataextractor.scrape.clean.Cleaner object>

clean_abstract = <chemdataextractor.text.processors.Chain object>

clean_headings = <chemdataextractor.scrape.clean.Cleaner object>

clean_paragraphs = <chemdataextractor.text.processors.Chain object>

process_abstract = <chemdataextractor.text.normalize.Normalizer object>

process_headings = <chemdataextractor.text.normalize.Normalizer object>

process_paragraphs = <chemdataextractor.text.processors.Chain object>

process_license = <chemdataextractor.text.processors.Chain object>

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'accepted

.scrape.pub.elsevier

Tools for scraping documents from Elsevier.

copyright Copyright 2017 by Callum Court.

license MIT, see LICENSE file for more details.

chemdataextractor.scrape.pub.elsevier.**CHAR_REPLACEMENTS = [('\\[?\\[1 with combining macro**
Map placeholder text to unicode characters.

chemdataextractor.scrape.pub.elsevier.**elsevier_substitute = <chemdataextractor.text.proces**
Substitutor that replaces ACS escape codes with the actual unicode character

class chemdataextractor.scrape.pub.elsevier.**ElsevierSearchDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Document information from Elsevier API search results.

test
A string field.

fields = {'test': <chemdataextractor.scrape.fields.StringField object>}

class chemdataextractor.scrape.pub.elsevier.**ElsevierSearchScraper**
Bases: *chemdataextractor.scrape.scraper.UrlScraper*

Scraper for Elsevier search results.

entity
alias of *ElsevierSearchDocument*

make_request(*url*)
Make a HTTP GET request.

Parameters **url** – The URL to get.

Returns The response to the request.

Return type requests.Response

run(*url*)
Request URL, scrape response and return an EntityList.

class chemdataextractor.scrape.pub.elsevier.**ElsevierImage**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Embedded figure. Includes both Schemes and Figures.

caption
A string field.

image_url
A string field.

process_caption = <chemdataextractor.text.processors.Chain object>

process_image_url = <chemdataextractor.text.processors.LAdd object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'image_url

class chemdataextractor.scrape.pub.elsevier.**ElsevierTableData**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Embedded row data from document tables

rows
A string field.

fields = {'rows': <chemdataextractor.scrape.fields.StringField object>}

class chemdataextractor.scrape.pub.elsevier.**ElsevierTable**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Table within document.

title
A string field.

column_headings
A string field.

data
A field that contains another Entity.

caption
A string field.

process_title = <chemdataextractor.text.processors.Chain object>

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'column_he

class chemdataextractor.scrape.pub.elsevier.**ElsevierHtmlDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.DocumentEntity*

Scraper of document information from Elsevier html papers

doi
A string field.

title
A string field.

authors
A string field.

abstract
A string field.

journal
A string field.

volume
A string field.

copyright
A string field.

headings
A string field.

sub_headings
A string field.

html_url
A field with optional URL processing.

paragraphs
A string field.

figures
A field that contains another Entity.

published_date
A string field.

citations
A string field.

tables
A field that contains another Entity.

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'authors'

class chemdataextractor.scrape.pub.elsevier.**ElsevierHtmlScraper**
Bases: *chemdataextractor.scrape.scraper.UrlScraper*

Scraper for Elsever html paper pages

entity
alias of *ElsevierHtmlDocument*

class chemdataextractor.scrape.pub.elsevier.**ElsevierXmlImage**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

caption
A string field.

label
A string field.

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'label':

class chemdataextractor.scrape.pub.elsevier.**ElsevierXmlTableData**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

rows
A string field.

fields = {'rows': <chemdataextractor.scrape.fields.StringField object>}

class chemdataextractor.scrape.pub.elsevier.**ElsevierXmlTable**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

label
A string field.

caption
A string field.

column_headings
A field that contains another Entity.

data
A field that contains another Entity.

fields = {'caption': <chemdataextractor.scrape.fields.StringField object>, 'column_he

class chemdataextractor.scrape.pub.elsevier.**ElsevierXmlDocument**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Scraper for Elsevier XML articles

doi
A string field.

title
A string field.

authors
A string field.

abstract
A string field.

journal
A string field.

volume
A string field.

issue
A string field.

pages
A string field.

firstpage
A string field.

lastpage
A string field.

copyright
A string field.

publisher
A string field.

headings
A string field.

url
A field with optional URL processing.

paragraphs
A string field.

figures
A field that contains another Entity.

published_date
A string field.

citations
A string field.

tables
A field that contains another Entity.

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'authors'

process_abstract = <chemdataextractor.text.processors.Chain object>

3.11.2 .scrape.base

Abstract base classes that define the interface for Scrapers, Fields, Crawlers, etc.

class chemdataextractor.scrape.base.**BaseScraper**
Bases: object

Abstract Scraper class from which all Scrapers inherit.

root = None
CSS selector or XPath expression that returns the root of each entity.

root_xpath = False
Whether the root is an XPath expression instead of a CSS selector.

__init__()

create_session()
Override to set up default data (e.g. headers, authentication) on each request.

name()
A unique name for this scraper.

entity
The Entity to scrape.

process_entity(*entity*)
Override to process each entity.

make_request(*url*, *data*)
Make a HTTP request.

Parameters

- **url** – The URL to get.
- **data** – Query data.

Returns The response to the request.

Return type requests.Response

process_response(*response*)
Return a Selector for the given response.

Parameters **response** (*requests.Response*) – The response object.

Return type *Selector*

get_roots(*selector*)

class chemdataextractor.scrape.base.**BaseFormat**
Bases: object

process_response(*response*)
Return a Selector for the given response.

Parameters **response** (*requests.Response*) – The response object.

Return type *Selector*

class chemdataextractor.scrape.base.**BaseRequester**
Bases: object

make_request(*url*, *data*)
Make a HTTP request.

Parameters

- **url** – The URL to get.
- **data** – Query data.

Returns The response to the request.

Return type requests.Response

class chemdataextractor.scrape.base.**BaseEntityProcessor**
Bases: object

Abstract EntityProcessor class from which all EntityProcessors inherit.

process_entity(*entity*)
Process an Entity. Return None to filter Entity from the pipeline.

Parameters **entity** (chemdataextractor.scrape.entity.Entity) – The Entity to process.

Returns The processed Entity.

Return type *Entity* or None

class chemdataextractor.scrape.base.**BaseEntity**
Bases: object

Abstract Entity class from which all Entities inherit.

class chemdataextractor.scrape.base.**EntityMeta**
Bases: abc.ABCMeta

Metaclass for Entity.

class chemdataextractor.scrape.base.**BaseField**(*selection*, *xpath=False*, *re=None*, *all=False*, *default=None*, *null=False*, *raw=False*)
Bases: object

Base class for all fields.

name = None

__init__(*selection*, *xpath=False*, *re=None*, *all=False*, *default=None*, *null=False*, *raw=False*)

Parameters

- **selection** (*string*) – The CSS selector or XPath expression used to select the content to scrape.

- **xpath** (*bool*) – (Optional) Whether selection is an XPath expression instead of a CSS selector. Default False.
- **re** – (Optional) Regular expression to apply to scraped content.
- **all** (*bool*) – (Optional) Whether to scrape all occurrences instead of just the first. Default False.
- **default** – (Optional) The default value for this field if none is set.
- **null** (*bool*) – (Optional) Include in serialized output even if value is None. Default False.
- **raw** (*bool*) – (Optional) Whether to scrape the raw HTML/XML instead of the text contents. Default False.

scrape(*selector*, *cleaner=None*, *processor=None*)
Scrape the value for this field from the selector.

serialize(*value*)
Serialize this field.

process(*value*)
Override to perform custom processing of a value.

3.11.3 .scrape.clean

Tools for cleaning up XML/HTML by removing tags entirely or replacing with their contents.

class chemdataextractor.scrape.clean.**Cleaner**(***kwargs*)
Bases: object

Clean HTML or XML by removing tags completely or replacing with their contents.

A Cleaner instance provides a clean_markup method:

```
cleaner = Cleaner()
htmlstring = '<html><body><script>alert("test")</script><p>Some text</p></body></
↪html>'
print(cleaner.clean_markup(htmlstring))
```

A Cleaner instance is also a callable that can be applied to lxml document trees:

```
tree = lxml.etree.fromstring(htmlstring)
cleaner(tree)
print(lxml.etree.tostring(tree))
```

Elements that are matched by kill_xpath are removed entirely, along with their contents. By default, kill_xpath matches all script and style tags, as well as comments and processing instructions.

Elements that are matched by strip_xpath are replaced with their contents. By default, no elements are stripped. A common use-case is to set strip_xpath to .//*, which specifies that all elements should be stripped.

Elements that are matched by allow_xpath are excepted from stripping, even if they are also matched by strip_xpath. This is useful when setting strip_xpath to strip all tags, allowing a few expections to be specified by allow_xpath.

kill_xpath = './/script | .//style | .//comment() | .//processing-instruction() | .//*

strip_xpath = None

allow_xpath = None

fix_whitespace = True

namespaces = {'dc': 'http://purl.org/dc/elements/1.1/', 'prism': 'http://prismstanda

__init__(***kwargs*)
Behaviour can be customized by overriding attributes in a subclass or setting them in the constructor.

Parameters

- **kill_xpath** (*string*) – XPath expression for tags to remove along with their contents.
- **strip_xpath** (*string*) – XPath expression for tags to replace with their contents.
- **allow_xpath** (*string*) – XPath expression for tags to except from strip_xpath.
- **fix_whitespace** (*bool*) – Normalize whitespace to a single space and ensure newlines around block elements.
- **namespaces** (*dict*) – Namespace prefixes to register for the XPaths.

clean_html(*html*)
Apply `Cleaner` to HTML string or document and return a cleaned string or document.

clean_markup(*markup*, *parser=None*)
Apply `Cleaner` to markup string or document and return a cleaned string or document.

chemdataextractor.scrape.clean.**clean = <chemdataextractor.scrape.clean.Cleaner object>**
A default Cleaner instance, which kills comments, processing instructions, script tags, style tags.

chemdataextractor.scrape.clean.**clean_markup = <bound method Cleaner.clean_markup of <chemda**
Convenience function for applying `clean` to a string.

chemdataextractor.scrape.clean.**clean_html = <bound method Cleaner.clean_html of <chemdatae**
Convenience function for applying `clean` to a HTML string.

chemdataextractor.scrape.clean.**strip = <chemdataextractor.scrape.clean.Cleaner object>**
A Cleaner instance that is configured to strip all tags, replacing them with their text contents.

chemdataextractor.scrape.clean.**strip_markup = <bound method Cleaner.clean_markup of <chemda**
Convenience function for applying `strip` to a string.

chemdataextractor.scrape.clean.**strip_html = <bound method Cleaner.clean_html of <chemdatae**
Convenience function for applying `strip` to a HTML string.

3.11.4 .scrape.csstranslator

Extend cssselect to improve handling of pseudo-elements.

This is derived from csstranslator.py in the Scrapy project. The original file is available at: https://github.com/scrapy/scrapy/blob/master/scrapy/selector/csstranslator.py

The original file was released under the BSD license:

Copyright (c) Scrapy developers. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of Scrapy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

class chemdataextractor.scrape.csstranslator.**CdeXPathExpr**(*path=''*, *element='*'*, *condition=''*, *star_prefix=False*)

Bases: cssselect.xpath.XPathExpr

textnode = False

attribute = None

classmethod from_xpath(*xpath*, *textnode=False*, *attribute=None*)

join(*combiner*, *other*)

class chemdataextractor.scrape.csstranslator.**TranslatorMixin**

Bases: object

xpath_element(*selector*)

xpath_pseudo_element(*xpath*, *pseudo_element*)

xpath_attr_functional_pseudo_element(*xpath*, *function*)

xpath_text_simple_pseudo_element(*xpath*)

Support selecting text nodes using ::text pseudo-element

class chemdataextractor.scrape.csstranslator.**CssXmlTranslator**

Bases: *chemdataextractor.scrape.csstranslator.TranslatorMixin*, cssselect.xpath.GenericTranslator

class chemdataextractor.scrape.csstranslator.**CssHTMLTranslator**(*xhtml=False*)

Bases: *chemdataextractor.scrape.csstranslator.TranslatorMixin*, cssselect.xpath.HTMLTranslator

3.11.5 .scrape.entity

An entity to extract.

class chemdataextractor.scrape.entity.**Entity**(*selector*)

Bases: *chemdataextractor.scrape.base.BaseEntity*

fields = {}

__init__(*selector*)

Parameters **selector** (Selector) – The selector to scrape.

classmethod scrape(*selector*, *root*, *xpath=False*)

Return EntityList for the given selector.

serialize()
Convert Entity to python dictionary.

to_json(**args*, ***kwargs*)
Convert Entity to JSON.

class chemdataextractor.scrape.entity.**EntityList**(**entities*)
Bases: collections.abc.Sequence

Wrapper around a list of Entities to facilitate operations on all at once.

__init__(**entities*)
Initialize self. See help(type(self)) for accurate signature.

serialize()
Serialize to a list of python dictionaries.

to_json(**args*, ***kwargs*)
Convert EntityList to JSON.

class chemdataextractor.scrape.entity.**DocumentEntity**(*selector*)
Bases: *chemdataextractor.scrape.entity.Entity*

Generic document entity.

doi
A string field.

title
A string field.

authors
A string field.

published_date
A datetime field. Depends on python-dateutil.

online_date
A datetime field. Depends on python-dateutil.

journal
A string field.

volume
A string field.

issue
A string field.

firstpage
A string field.

lastpage
A string field.

abstract
A string field.

publisher
A string field.

issn
A string field.

language
A string field.

copyright
A string field.

license
A field with optional URL processing.

html_url
A field with optional URL processing.

pdf_url
A field with optional URL processing.

landing_url
A field with optional URL processing.

process_title = <chemdataextractor.text.normalize.Normalizer object>

process_journal = <chemdataextractor.text.normalize.Normalizer object>

process_publisher = <chemdataextractor.text.normalize.Normalizer object>

process_authors = <chemdataextractor.text.normalize.Normalizer object>

process_abstract = <chemdataextractor.text.normalize.Normalizer object>

fields = {'abstract': <chemdataextractor.scrape.fields.StringField object>, 'authors'

3.11.6 .scrape.fields

Fields to define on an entity.

class chemdataextractor.scrape.fields.**StringField**(*selection*, *lower=False*, *upper=False*, *strip=False*, ***kwargs*)

Bases: *chemdataextractor.scrape.base.BaseField*

A string field.

__init__(*selection*, *lower=False*, *upper=False*, *strip=False*, ***kwargs*)

Parameters

- **lower** (*bool*) – (Optional) Whether to lowercase the string. Default False.
- **upper** (*bool*) – (Optional) Whether to uppercase the string. Default False.
- **strip** (*bool*) – (Optional) Whether to strip whitespace from start/end. Default False.

process(*value*)
Override to perform custom processing of a value.

class chemdataextractor.scrape.fields.**UrlField**(*selection*, *strip_querystring=False*, ***kwargs*)

Bases: *chemdataextractor.scrape.fields.StringField*

A field with optional URL processing.

__init__(*selection*, *strip_querystring=False*, ***kwargs*)

Parameters **strip_querystring** – (Optional) Whether to remove the querystring. Default False.

process(*value*)
Override to perform custom processing of a value.

class chemdataextractor.scrape.fields.**EntityField**(*entity*, *selection*, ***kwargs*)
Bases: *chemdataextractor.scrape.base.BaseField*

A field that contains another Entity.

__init__(*entity*, *selection*, ***kwargs*)

Parameters **entity** – The embedded entity.

scrape(*selector*, *cleaner=None*, *processor=None*)
Scrape the value for this field from the selector.

class chemdataextractor.scrape.fields.**IntField**(*selection*, *xpath=False*, *re=None*, *all=False*, *default=None*, *null=False*, *raw=False*)
Bases: *chemdataextractor.scrape.base.BaseField*

An integer number field.

process(*value*)
Convert value to an int.

class chemdataextractor.scrape.fields.**FloatField**(*selection*, *xpath=False*, *re=None*, *all=False*, *default=None*, *null=False*, *raw=False*)
Bases: *chemdataextractor.scrape.base.BaseField*

An floating point number field.

process(*value*)
Convert value to a float.

class chemdataextractor.scrape.fields.**BoolField**(*selection*, *true=re.compile('true|yes|1'*, *re.IGNORECASE)*, *false=re.compile('false|no|0'*, *re.IGNORECASE)*, ***kwargs*)
Bases: *chemdataextractor.scrape.base.BaseField*

A boolean field type.

__init__(*selection*, *true=re.compile('true|yes|1'*, *re.IGNORECASE)*, *false=re.compile('false|no|0'*, *re.IGNORECASE)*, ***kwargs*)

Parameters

- **true** – Regular expression match that evaluates to True.
- **false** – Regular expression match that evaluates to False.

process(*value*)
Override to perform custom processing of a value.

class chemdataextractor.scrape.fields.**DateTimeField**(*selection*, *xpath=False*, *re=None*, *all=False*, *default=None*, *null=False*, *raw=False*)
Bases: *chemdataextractor.scrape.base.BaseField*

A datetime field. Depends on python-dateutil.

process(*value*)
Override to perform custom processing of a value.

serialize(*value*)
Serialize this field.

3.11.7 .scrape.scraper

Concrete classes for scraping and searching.

class chemdataextractor.scrape.scraper.**HtmlFormat**
Bases: *chemdataextractor.scrape.base.BaseFormat*

Process HTML response and return a Selector.

process_response(*response*)
Return a Selector for the given response.

Parameters **response** (*requests.Response*) – The response object.

Return type *Selector*

class chemdataextractor.scrape.scraper.**XmlFormat**
Bases: *chemdataextractor.scrape.base.BaseFormat*

Process XML response and return a Selector.

namespaces = None

process_response(*response*)
Return a Selector for the given response.

Parameters **response** (*requests.Response*) – The response object.

Return type *Selector*

class chemdataextractor.scrape.scraper.**GetRequester**
Bases: *chemdataextractor.scrape.base.BaseRequester*

make_request(*session*, *url*, ***kwargs*)
Make a HTTP GET request.

Parameters **url** – The URL to get.

Returns The response to the request.

Return type requests.Response

class chemdataextractor.scrape.scraper.**PostRequester**
Bases: *chemdataextractor.scrape.base.BaseRequester*

make_request(*session*, *url*, ***kwargs*)
Make a HTTP POST request.

Parameters

- **url** – The URL to post to.
- **data** – The data to post.

Returns The response to the request.

Return type requests.Response

class chemdataextractor.scrape.scraper.**UrlScraper**
Bases: *chemdataextractor.scrape.scraper.GetRequester*, *chemdataextractor.scrape.scraper.HtmlFormat*, *chemdataextractor.scrape.base.BaseScraper*

Scraper that takes a URL as input.

process_url(*url*)
Override to filter or process input URL prior to making request.

run(*url*)
Request URL, scrape response and return an EntityList.

class chemdataextractor.scrape.scraper.**RssScraper**
Bases: *chemdataextractor.scrape.scraper.XmlFormat*, *chemdataextractor.scrape.scraper.UrlScraper*

RSS scraper

root = 'item'

namespaces = {'atom': 'http://www.w3.org/2005/Atom', 'feedburner': 'http://rssnamesp

class chemdataextractor.scrape.scraper.**SearchScraper**
Bases: *chemdataextractor.scrape.scraper.GetRequester*, *chemdataextractor.scrape.scraper.HtmlFormat*, *chemdataextractor.scrape.base.BaseScraper*

Scraper that takes a search query as input.

process_query(*query*)
Override to filter or process input query prior to making request.

perform_search(*query*, *page*)
Override to implement search. Take query input and return a SearchResult.

run(*query*, *page=1*)

class chemdataextractor.scrape.scraper.**SearchResult**
Bases: object

Class to handle results from a search query to websites, regardless of method of scraping used.

selector
Process the result of the search, giving a selector

Returns The result of the search

Return type selector

class chemdataextractor.scrape.scraper.**SeleniumSearchResult**(*driver*)
Bases: object

Search results when using Selenium for scraping

__init__(*driver*)

Parameters **driver** (*selenium.webdriver*) – driver from which results will be scraped.

selector

class chemdataextractor.scrape.scraper.**ResponseSearchResult**(*response*)
Bases: object

Search results when using the requests library for scraping

__init__(*response*)

Parameters **response** (*requests.Response*) – HTML response for results

selector

3.11.8 .scrape.selector

Tool for selecting content from HTML or XML using CSS or XPath expressions.

class chemdataextractor.scrape.selector.**Selector**(*root*, *fmt='html'*, *translator=<class 'chemdataextractor.scrape.csstranslator.CssHTMLTranslator'>*, *namespaces=None*)

Bases: object

Tool for selecting content from HTML or XML using XPath selectors.

__init__(*root*, *fmt='html'*, *translator=<class 'chemdataextractor.scrape.csstranslator.CssHTMLTranslator'>*, *namespaces=None*)
Initialize self. See help(type(self)) for accurate signature.

classmethod from_text(*text*, *base_url=None*, *parser=<class 'lxml.html.HTMLParser'>*, *translator=<class 'chemdataextractor.scrape.csstranslator.CssHTMLTranslator'>*, *fmt='html'*, *namespaces=None*, *encoding=None*)

classmethod from_html_text(*text*, *base_url=None*, *namespaces=None*, *encoding=None*)

classmethod from_xml_text(*text*, *base_url=None*, *namespaces=None*, *encoding=None*)

classmethod from_response(*response*, *parser=<class 'lxml.html.HTMLParser'>*, *translator=<class 'chemdataextractor.scrape.csstranslator.CssHTMLTranslator'>*, *fmt='html'*, *namespaces=None*)

classmethod from_html(*response*, *namespaces=None*)

classmethod from_xml(*response*, *namespaces=None*)

path
Absolute path to the root of this selector.

tag
Tag name of the root of this selector.

xpath(*query*)

css(*query*)

re(*regex*)

extract(*cleaner=None*, *raw=False*)

class chemdataextractor.scrape.selector.**SelectorList**(**selectors*)

Bases: collections.abc.Sequence

Wrapper around a list of Selectors to allow selecting from all at once.

__init__(**selectors*)
Initialize self. See help(type(self)) for accurate signature.

xpath(*xpath*)

re(*regex*)

extract(*cleaner=None*, *raw=False*)

extract_first(*cleaner=None*, *raw=False*, *default=None*)

3.12 .text

Useful tools for processing text

Tools for processing text.

chemdataextractor.text.**CONTROLS = {'\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07',**
Control characters.

chemdataextractor.text.**HYPHENS = {'-', '‐', '‑', '‒', '–', '—', '―', ''}**
Hyphen and dash characters.

chemdataextractor.text.**MINUSES = {'−', '', '', ''}**
Minus characters.

chemdataextractor.text.**PLUSES = {'+', '', ''}**
Plus characters.

chemdataextractor.text.**SLASHES = {'/', '⁄', ''}**
Slash characters.

chemdataextractor.text.**TILDES = {'~', '˜', '', '', '', '', '', ''}**
Tilde characters.

chemdataextractor.text.**APOSTROPHES = {"'", '', '’', '', '', ''}**
Apostrophe characters.

chemdataextractor.text.**SINGLE_QUOTES = {"'", '‘', '’', '‚', ''}**
Single quote characters.

chemdataextractor.text.**DOUBLE_QUOTES = {'"', '“', '”', '„', ''}**
Double quote characters.

chemdataextractor.text.**ACCENTS = {'`', '´'}**
Accent characters.

chemdataextractor.text.**PRIMES = {'', '', '', '', '', '', ''}**
Prime characters.

chemdataextractor.text.**QUOTES = {'"', "'", '`', '´', '', '‘', '’', ',', '', '“', '”', '„',**
Quote characters, including apostrophes, single quotes, double quotes, accents and primes.

chemdataextractor.text.**GREEK = {'A', 'B', 'Γ', 'Δ', 'E', 'Z', 'H', 'Θ', 'I', 'K', 'Λ', 'M'**
Uppercase and lowercase greek letters.

chemdataextractor.text.**GREEK_WORDS = {'Alpha', 'Beta', 'Chi', 'Delta', 'Epsilon', 'Eta', 'G**
Names of greek letters spelled out as words.

chemdataextractor.text.**SMALL = {'a', 'an', 'and', 'as', 'at', 'but', 'by', 'en', 'for', 'if**
Words that should not be capitalized in titles.

chemdataextractor.text.**NAME_SMALL = {'abu', 'bin', 'bon', 'da', 'dal', 'de', 'del', 'der',**
Words that should not be capitalized in names.

chemdataextractor.text.**NUMBERS = {'billion', 'eight', 'eighteen', 'eighty', 'eleven', 'fift**
A variety of numbers, spelled out as words.

chemdataextractor.text.**EMAIL_RE = re.compile('([\\w\\-\\.\\+%]+@(\\w[\\w\\-]+\\.)+[\\w\\-]+**
Regular expression that matches email addresses.

chemdataextractor.text.**DOI_RE = re.compile('^10\\.\\d{4,9}/[-\\._;()/:A-Z0-9]+$')**
Regular expression that matches DOIs.

chemdataextractor.text.**ISSN_RE = re.compile('^\\d{4}-\\d{3}[\\dX]$')**
Regular expression that matches ISSNs.

chemdataextractor.text.**CONTROL_RE = re.compile('[^ -\ud7ff\t\n\r\ue000-0-FF]+')**
Regular expression that matches control characters not allowed in XML.

chemdataextractor.text.**get_encoding**(*input_string*, *guesses=None*, *is_html=False*)
Return the encoding of a byte string. Uses bs4 UnicodeDammit.

Parameters

- **input_string** (*string*) – Encoded byte string.
- **guesses** (*list[string]*) – (Optional) List of encoding guesses to prioritize. Default is ['utf-8']
- **is_html** (*bool*) – Whether the input is HTML.

chemdataextractor.text.**levenshtein**(*s1*, *s2*, *allow_substring=False*)
Return the Levenshtein distance between two strings.

The Levenshtein distance (a.k.a "edit difference") is the number of characters that need to be substituted, inserted or deleted to transform s1 into s2.

Setting the *allow_substring* parameter to True allows s1 to be a substring of s2, so that, for example, "hello" and "hello there" would have a distance of zero.

Parameters

- **s1** (*string*) – The first string
- **s2** (*string*) – The second string
- **allow_substring** (*bool*) – Whether to allow s1 to be a substring of s2

Returns Levenshtein distance.

Type int

chemdataextractor.text.**bracket_level**(*text*, *open={'(', '[', '{'}*, *close={')', ']', '}'}*)
Return 0 if string contains balanced brackets or no brackets.

chemdataextractor.text.**is_punct**(*text*)

chemdataextractor.text.**is_ascii**(*text*)

chemdataextractor.text.**like_url**(*text*)

chemdataextractor.text.**like_number**(*text*)

chemdataextractor.text.**word_shape**(*text*)

3.12.1 .text.chem

Chemistry text handling tools.

chemdataextractor.text.chem.**extract_inchis**(*s*)
Return a list of InChI identifiers extracted from the string.

chemdataextractor.text.chem.**extract_inchikeys**(*s*)
Return a list of InChIKey identifiers extracted from the string.

chemdataextractor.text.chem.**extract_cas**(*s*)
Return a list of CAS identifiers extracted from the string.

chemdataextractor.text.chem.**extract_smiles**(*s*)
Return a list of SMILES identifiers extracted from the string.

3.12.2 .text.latex

Tools for converting LaTeX to unicode.

chemdataextractor.text.latex.**latex_to_unicode**(*text*, *capitalize=False*)
Replace LaTeX entities with the equivalent unicode and optionally capitalize.

Parameters

- **text** – The LaTeX string to be converted
- **capitalize** – Can be 'sentence', 'name', 'title', 'upper', 'lower'

3.12.3 .text.normalize

Tools for normalizing text.

class chemdataextractor.text.normalize.**BaseNormalizer**
Bases: *chemdataextractor.text.processors.BaseProcessor*

Abstract normalizer class from which all normalizers inherit.

Subclasses must implement a `normalize()` method.

normalize(*text*)
Normalize the text.

Parameters **text** (*string*) – The text to normalize.

Returns Normalized text.

Return type string

class chemdataextractor.text.normalize.**Normalizer**(*form='NFKC'*, *strip=True*, *collapse=True*, *hyphens=False*, *quotes=False*, *ellipsis=False*, *slashes=False*, *tildes=False*)
Bases: *chemdataextractor.text.normalize.BaseNormalizer*

Main Normalizer class for generic English text.

Normalize unicode, hyphens, quotes, whitespace.

By default, the normal form NFKC is used for unicode normalization. This applies a compatibility decomposition, under which equivalent characters are unified, followed by a canonical composition. See Python docs for information on normal forms: http://docs.python.org/2/library/unicodedata.html#unicodedata.normalize

__init__(*form='NFKC'*, *strip=True*, *collapse=True*, *hyphens=False*, *quotes=False*, *ellipsis=False*, *slashes=False*, *tildes=False*)

Parameters

- **form** (*string*) – Normal form for unicode normalization.
- **strip** (*bool*) – Whether to strip whitespace from start and end.
- **collapse** (*bool*) – Whether to collapse all whitespace (tabs, newlines) down to single spaces.

- **hyphens** (*bool*) – Whether to normalize all hyphens, minuses and dashes to the ASCII hyphen-minus character.
- **quotes** (*bool*) – Whether to normalize all apostrophes, quotes and primes to the ASCII quote character.
- **ellipsis** (*bool*) – Whether to normalize ellipses to three full stops.
- **slashes** (*bool*) – Whether to normalize slash characters to the ASCII slash character.
- **tildes** (*bool*) – Whether to normalize tilde characters to the ASCII tilde character.

normalize(*text*)
Run the Normalizer on a string.

Parameters **text** – The string to normalize.

chemdataextractor.text.normalize.**normalize = <chemdataextractor.text.normalize.Normalizer**
Default normalize that canonicalizes unicode and fixes whitespace.

chemdataextractor.text.normalize.**strict_normalize = <chemdataextractor.text.normalize.Norma**
More aggressive normalize that also standardizes hyphens, and quotes.

class chemdataextractor.text.normalize.**ExcessNormalizer**(*form='NFKC'*, *strip=True*, *collapse=True*, *hyphens=True*, *quotes=True*, *ellipsis=True*, *tildes=True*)
Bases: *chemdataextractor.text.normalize.Normalizer*

Excessive string normalization.

This is useful when doing fuzzy string comparisons. A common use case is to run this before calculating the Levenshtein distance between two strings, so that only "important" differences are counted.

__init__(*form='NFKC'*, *strip=True*, *collapse=True*, *hyphens=True*, *quotes=True*, *ellipsis=True*, *tildes=True*)

normalize(*text*)
Run the Normalizer on a string.

Parameters **text** – The string to normalize.

class chemdataextractor.text.normalize.**ChemNormalizer**(*form='NFKC'*, *strip=True*, *collapse=True*, *hyphens=True*, *quotes=True*, *ellipsis=True*, *tildes=True*, *chem_spell=True*)
Bases: *chemdataextractor.text.normalize.Normalizer*

Normalizer that also unifies chemical spelling.

__init__(*form='NFKC'*, *strip=True*, *collapse=True*, *hyphens=True*, *quotes=True*, *ellipsis=True*, *tildes=True*, *chem_spell=True*)

normalize(*text*)
Normalize unicode, hyphens, whitespace, and some chemistry terms and formatting.

3.12.4 .text.processors

Text processors.

class chemdataextractor.text.processors.**BaseProcessor**
Bases: object

Abstract processor class from which all processors inherit. Subclasses must implement a __call__() method.

class chemdataextractor.text.processors.**Chain**(**callables*)
Bases: object

Apply a series of processors in turn. Stops if a processors returns None.

__init__(**callables*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.text.processors.**Discard**(**match*)
Bases: object

Return None if value matches a string.

__init__(**match*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.text.processors.**LAdd**(*substring*)
Bases: object

Add a substring to the start of a value.

__init__(*substring*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.text.processors.**RAdd**(*substring*)
Bases: object

Add a substring to the end of a value.

__init__(*substring*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.text.processors.**LStrip**(**substrings*)
Bases: object

Remove a substring from the start of a value.

__init__(**substrings*)
Initialize self. See help(type(self)) for accurate signature.

class chemdataextractor.text.processors.**RStrip**(**substrings*)
Bases: object

Remove a substring from the end of a value.

__init__(**substrings*)
Initialize self. See help(type(self)) for accurate signature.

chemdataextractor.text.processors.**floats**(*s*)
Convert string to float. Handles more string formats that the standard python conversion.

chemdataextractor.text.processors.**strip_querystring**(*url*)
Remove the querystring from the end of a URL.

class chemdataextractor.text.processors.**Substitutor**(*substitutions*)
Bases: object

Perform a list of substitutions defined by regex on text.

Useful to clean up text where placeholders are used in place of actual unicode characters.

__init__(*substitutions*)

Parameters **substitutions** – List of (regex, string) tuples that define the substitution.

chemdataextractor.text.processors.**extract_emails**(*text*)
Return a list of email addresses extracted from the string.

chemdataextractor.text.processors.**unapostrophe**(*text*)
Strip apostrophe and 's' from the end of a string.

config	Config file reader/writer.
data	
errors	Error classes for ChemDataExtractor.
utils	Miscellaneous utility functions.

biblio
biblio.bibtex
biblio.person
biblio.xmp

cli
cli.cem
cli.chemdner
cli.cluster
cli.config
cli.data
cli.dict
cli.evaluate
cli.pos
cli.tokenize

doc
doc.document
doc.element
doc.figure
doc.meta
doc.table
doc.text

eval	Evaluation of extraction results
eval.evaluation	

model

Continued on next page

Table 6 – continued from previous page

model.base
model.model
model.units
model.units.unit
model.units.dimension
model.units.quantity_model
model.units.length
model.units.mass
model.units.time
model.units.temperature

nlp
nlp.abbrev
nlp.cem
nlp.corpus
nlp.lexicon
nlp.pos
nlp.tag
nlp.tokenize

parse
parse.actions
parse.auto
parse.base
parse.cem
parse.common
parse.context
parse.elements
parse.ir
parse.mp
parse.nmr
parse.tg
parse.uvvis

reader
reader.acs
reader.base
reader.cssp
reader.markup
reader.nlm
reader.pdf
reader.plaintext
reader.rsc
reader.uspto
reader.elsevier

Continued on next page

Table 9 – continued from previous page

reader.springer

relex
relex.cluster
relex.entity
relex.pattern
relex.phrase
relex.relationship
relex.snowball
relex.utils

scrape
scrape.base
scrape.clean
scrape.csstranslator
scrape.entity
scrape.fields
scrape.scraper
scrape.selector
scrape.pub
scrape.pub.nlm
scrape.pub.rsc
scrape.pub.springer
scrape.pub.elsevier

text	Tools for processing text.
text.chem	Chemistry text handling tools.
text.latex	Tools for converting LaTeX to unicode.
text.normalize	Tools for normalizing text.
text.processors	Text processors.

CHAPTER 4

v2.0 Migration Guide

4.1 Overview

Previously, ChemDataExtractor required huge amounts of manual input to create parsers for new models and if the user had multiple new models that were similar, we ended up with huge amounts of duplicated code. This system was very user-unfriendly and didn't really follow the logic of how Physics and Chemistry are structured.

Furthermore, the Interdependency Resolution (IR) for these models was purely backwards-looking and did not account for dynamic or document-specific terminology even when it was clearly defined in the text.

To resolve some of these issues we have developed a new model-based approach to parsing that intrinsically links the models and the parsers, enabling completely automated parser creation and forward-looking Interdependency Resolution.

This has resulted in some breaking changes to the API for ChemDataExtractor, however, these are all fixable with minor changes to existing code, which are outlined in the section *Migrating Existing Code*.

4.2 Changes to ChemDataExtractor

4.2.1 Overall Structure

At a high level, in previous versions of ChemDataExtractor, the *Document* class and each of its subelements (e.g. *Paragraph*, *Table* or *Sentence*) had a list of parsers. These parsers each had an associated model which they were parsing for. When these parsers found a sentence (or table cell) that matched to the parse phrase root, it would create a `Compound` and the property would be associated to this instance of a compound.

The new structure changes this hierarchy significantly. The *Document* class and its subelements now own the models that they should look for. Each model contains a list of parsers that can be used for parsing different types of elements (e.g. *Sentence* or *Table*) to extract the model. At the appropriate timings, the elements will call the appropriate parsers in the models.

This new structure has several advantages:

- You no longer have to search for the appropriate classes for parsing. You don't need to find `MpParser` and `MpTableParser` and assign them as parsers to *`Sentence`* s and *`Table`* s respectively to extract a *`MeltingPoint`*. With the new structure, you just pass in a list, `[MeltingPoint, Compound]`, to document, and the appropriate parsers are automatically used.
- The new structure is far safer, that is, it is impossible to use a parser meant for tables on a sentence and a parser meant for sentences on tables.
- The properties are no longer necessarily tied to the `Compound` class, meaning one could use ChemDataExtractor for other purposes too, such as extracting the conditions under which an experiment was done.
- You can easily build nested model hierarchies that more closely resemble the structure of Physics and Chemistry.

4.2.2 Changes to Models

In addition to the overall change of structure, involving each property optionally owning a *`Compound`*, new types of models have been introduced for the majority usecase of extracting a physical quantity structure, i.e. the case with a specifier, a value, and units, such as melting points, interatomic distances, and cooling rates. These models are all defined as subclasses of a new type of model, *`QuantityModel`*.

Note: While new Quantity-based models have been added to ChemDataExtractor, old-style models can still be used. Refer to the section *Migrating Existing Code* on how to have older models be extracted as similarly to the old behaviour as possible.

These model types can now be defined with minimal effort as the various base-quantities (Temperature, Length, Time etc) are included in ChemDataExtractor. Now for example, if we wished to create a new model that will be of type Temperature we simply inherit our model from the *`TemperatureModel`* class and define our entities.

Models of this type have only 2 requirements:

- A specifier with an associated parse expression (Optional, only required if autoparsers are desired). These parse expressions will be updated automatically using forward-looking Interdependency Resolution if the `updatable` flag is set to `True`.
- If applicable, a compound entity, named `compound`

While previous models in ChemDataExtractor stored values and units as strings, these are now automatically extracted and stored as numbers and *`Unit`* s, allowing for easy conversion and comparison. These changes are explored in more detail in *Addition of Units and Dimensions*.

Each entity must have a defined type, for example *`StringType`*, *`FloatType`* or *`ModelType`*. Note that by specifying *`ModelType`* you must provide another model, allowing for nested model relationships.

The entities also have properties:

- `parse_expression`: A `BaseElement` that is associated with the entity. This parse expression is used by the autoparser in constructing a parse rule.
- `required`: Whether or not the entity is required to form a relationship. If required is `True` and the entity is not found, the relationship will not be output by ChemDataExtractor.
- `contextual`: Whether or not the entity can be sourced from a different element to the rest of the entities, e.g. whether the entity can be completed with data from another sentence, or a different part of the table.
- `updatable`: Whether or not the `parse_expression` can be updated based on definitions found in the document (see *Forward looking Interdependency resolution*)

We can also add arbitrarily-named entities with any parse expressions we like. Example:

```
from models.units.temperature import TemperatureModel

#: My new model for finding Boiling points
class BoilingPoint(TemperatureModel):
    specifier = StringType(parse_expression=I('boiling')+I('point'),
                           required=True,
                           contextual=True,
                           updatable=True)
    compound = ModelType(Compound,
                         required=True,
                         contextual=False,
                         updatable=False)
    apparatus = ModelType(Apparatus, contextual=True)
    random_entity = StringType(parse_expression=I('complete')+I('nonsense'))
```

Notice also that we have added apparatus and compound as sub-models to `BoilingPoint`. If we pass in `BoilingPoint` to a document or a sentence, they will automatically also extract the apparatus and compound and associate them with the boiling point as required.

Model types for certain dimensions have not yet been defined. An example of how to create a new model is included in the Examples.

4.2.3 Addition of Units and Dimensions

Newly included in ChemDataExtractor are the concepts of *Unit* s and *Dimension* s. These work just as expected; each *Unit* has a dimension and quantities with the same *Unit* s can be converted between each other. See the API documentation for *chemdataextractor.model.units* for more information.

4.2.4 Changes to Parsers

Previously, different types of parsers were just distinguished by name. A `MpTableParser` was understood to parse tables, and `MpParser` was understood to parse sentences. However, this was not enforced in any way. This has now been changed, with all parsers now implementing either *parse_sentence()* if they are sentence parsers, or *parse_cell()* if a table parser. You can get these methods for free by subclassing from *BaseSentenceParser* and *BaseTableParser* respectively. You then only need to implement the interpret function, just as before. The role of the interpret function is identical to before, it takes a parse result and formats it to the desired model.

To work with the models now being able to store values and units in a more structured manner, *BaseParser* also now contains new methods for extracting them. Refer to the API documentation for more detail.

4.2.5 Forward looking Interdependency Resolution

More often than not, the specifier you define in your model will be insufficient for capturing all variations of the way in which the model is defined in text. In most cases, the specifier is given a short abbreviation such as:

"...the boiling point, bp,..."

Using the definition parsers within ChemDataExtractor, we now automatically update specifier entities at the document scope when these definitions are found. This means that the specifier parse-expression gets automatically updated to include the new definition. Following the example above, the new specifier parse expression will become:

```
(I('boiling') + I('point')) | I('bp')
```

Then for all remaining elements in the document, the relationship will be found if this specifier is used.

Note: This information only persists in the current document, so when a new document is parsed, we revert to the default defined specifier. This is to avoid the specifier parse expressions becoming too far removed from the original definition.

4.2.6 Integration with TableDataExtractor

TableDataExtractor is a new toolkit for ChemDataExtractor that vastly enhances its capabilities for information extraction from tabular data. Previously, rule-based parsers had to be written specifically for tables, for every new property. These would usually be very limited, due to the complexity of tables found in the literature.

TableDataExtractor reads all tables and outputs their data in a highly standardised format whilst also retaining information about all the row or column headings and subheadings that the data point belongs to. The output of TableDataExtractor is a *category table*, where each row corresponds to a single data-cell of the original table, along with its corresponding header structure. The standardized structure of the category table enables fully automated parsing with ChemDataExtractor. Within ChemDataExtractor all of the functionality of TableDataExtractor can be accessed via an instance of the *Table* object, `table`, as `table.tde_table`.

In most cases it should not be necessary to interact directly with TableDataExtractor. However, it is recommended to test it on an individual corpus of literature, before a production run. Visual inspection is the best option to do so:

```
from chemdataextractor import Document

f = open('my_dicument.xml', 'rb')
doc = Document.from_file(f)

for table in document.tables:
    table.tde_table.print_raw_table()
    print(table.tde_table)
```

This will print the raw table, as found in the source document (before processing with TableDataExtractor) as well as the structured category table, `table.tde_table`. For more information the following TableDataExtractor functionality:

- `print(table.tde_table.history)` will return information about the algorithms within TableDataExtractor that have been used on the particular table. If needed these can be tweaked by providing configuration parameters for TableDataExtractor (see TableDataExtractor documentation).
- `table.tde_table.print()` will print a more verbose output that includes the raw input table, the cleaned table (cleaned-up by TableDataExtractor), as well as a table that shows the labelling of the sections of the table.
- `table.tde_table.to_pandas()` outputs the table as Pandas DataFrame. This can be useful for further analysis.

More information can be found in the TableDataExtractor documentation.

4.2.7 Automatic Parsers

Due to the built-in forward-looking Interdependency Resolution we no longer have to manually specify as many specifiers when looking for new properties. The quantity extraction involving units and dimensions provides rich new metadata on our extracted values. These features make data extraction with ChemDataExtractor inherently much more powerful and context-rich.

We have taken advantage this new data to create automatic parsers for both sentences and tables. Any subclasses of *QuantityModel* have, by default, automatic parsers enabled, meaning no user intervention is needed to start extracting. These automatic parsers work especially well with the TableDataExtractor tables, which store the data in a highly standardised format. Thus, no user adjustments will be needed to extract data from tables.

Note: These parsers rely on the specifier and units information provided in *QuantityModel*, and described above. Therefore, they cannot be used with existing subclasses of *BaseModel* and, if needed, new model classes resembling the old ones can be written for that purpose.

4.2.8 Integration with Snowball

Due to the new ability of ChemDataExtractor to construct simple parsers automatically, Snowball is now fully integrated into the ChemDataExtractor workflow. Still, training of the Snowball algorithm needs to be performed. However, this is now much simpler to invoke. The Snowball algorithm is simply another parser that can optionally be used and can be passed into the models in the same way as any other custom created parser. Here is an example of using Snowball to extract Curie temperatures:

```
class CurieTemperature(TemperatureModel):
    specifier_expression = (I('Curie')+I('temperature') | I('TC')).add_action(join)
    specifier = StringType(parse_expression=specifier_expression, required=True,␣
↪contextual=False, updatable=True)
    compound = ModelType(Compound, required=True, contextual=True)


#1. Train from a single/multiple sentences/documents
s = Sentence('Cobalt displays a Curie Temperature of 1388 K which is higher than␣
↪BiFeO3.')
corpus = [s]

#2. Or train from a path to files
corpus = './tests/data/relex/curie_training_set/'

sb = Snowball(CurieTemperature)
sb.train(corpus)
CurieTemperature.parsers.append(sb)
```

4.2.9 Parsing

As a result we now have 3 different text parsing methods, each with its own advantages and disadvantages when it comes to extraction precision and recall.

The auto-generated text-parsers, of type *AutoSentenceParser* are very lenient. The root phrases for these parsers find any sentences that contain the required entities and return the first match to the models. As such, parsing with only the auto sentence parser will yield high recall but low precision. Furthermore, you will only extract correct relations from sentences that contain single instances of your model.

Snowball parsing is the opposite end of the precision-recall spectrum. Snowball is designed to be high precision and low recall based on the training data.

Therefore, if you wish to extract with both high precision and high recall, you will still need to write parse rules for complicated sentence structures, or train Snowball very extensively.

On the other hand, if you only wish to extract data from tables, the automated table parsers normally don't require any further adjustments for simple models.

4.3 Migrating Existing Code

This section is aimed at migrating existing code to run in ChemDataExtractor 2.0 without adding any new functionality. For information on how to take advantage of the new features please also refer to *Upgrading Existing Code*.

4.3.1 Migrating Models

When a model was previously written, a reference to the model would need to be added to Compound. This no longer needs to be done, so where the old version would have been:

```
from chemdataextractor.model import BaseModel, StringType, ListType, ModelType
from chemdataextractor.model import Compound

class BoilingPoint(BaseModel):
    value = StringType()
    units = StringType()

Compound.boiling_points = ListType(ModelType(BoilingPoint))
```

The new way to write this would be:

```
from chemdataextractor.model import BaseModel, StringType, ModelType
from chemdataextractor.model import Compound

class BoilingPoint(BaseModel):
    value = StringType()
    units = StringType()
    compound = ModelType(Compound)
    parsers = [BpParser()]
```

Where BpParser will be explained in the next section.

4.3.2 Migrating Parsers

The old way to write a parser would be to explicitly import the model and create it. This is no longer necessary, as all parsers contain a `model` attribute which is set at the required timing by the model. this means that a parser written before as:

```
import re
from chemdataextractor.parse import R, I, W, Optional, merge
from chemdataextractor.parse.base import BaseParser
from chemdataextractor.utils import first

prefix = (R(u'^b\.?p\.?$', re.I) | I(u'boiling') + I(u'point')).hide()
units = (W(u'°') + Optional(R(u'^[CFK]\.?$')))(u'units').add_action(merge)
value = R(u'^\d+(\.\d+)?$')(u'value')
bp = (prefix + value + units)(u'bp')

class BpParser(BaseParser):
    root = bp

    def interpret(self, result, start, end):
        compound = Compound(
            boiling_points=[
```

(continues on next page)

(continued from previous page)

```
                BoilingPoint(
                    value=first(result.xpath('./value/text()')),
                    units=first(result.xpath('./units/text()'))
                )
            ]
        )
        yield compound
```

would now be written as:

```
import re
from chemdataextractor.parse import R, I, W, Optional, merge
from chemdataextractor.parse.base import BaseSentenceParser
from chemdataextractor.utils import first
from chemdataextractor.model import Compound

prefix = (R(u'^b\.?p\.?$', re.I) | I(u'boiling') + I(u'point')).hide()
units = (W(u'°') + Optional(R(u'^[CFK]\.?$')))(u'units').add_action(merge)
value = R(u'^\d+(\.\d+)?$')(u'value')
bp = (prefix + value + units)(u'bp')

class BpParser(BaseSentenceParser):
    root = bp

    def interpret(self, result, start, end):
        boiling_point = self.model(value=first(result.xpath('./value/text()')),
                                   units=first(result.xpath('./units/text()')))
        yield boiling_point
```

Note also that the parser now inherits from *BaseSentenceParser* as opposed to *BaseParser* as it is a parser for sentences.

4.3.3 Extracting Properties

To extract a certain model, prior to 2.0, one had to set the parsers or the document. Instead of this, you now pass in the model that you want to extract from the document, so instead of this:

```
document.parsers = [BpParser()]
```

you would write:

```
document.models = [BoilingPoint]
```

Note that you should now pass in the class for the model you are parsing instead of an instance of the parser as before.

4.4 Upgrading Existing Code

The above small alterations are enough to get your code up and running, but to make the most of what ChemDataExtractor 2.0, you can upgrade your existing codebase to extract richer properties more easily.

4.4.1 Upgrading Models

A key new feature of version 2.0 are the new *QuantityModel* classes. These new models are much more versatile in that they extract values and errors as floats (or lists of floats), and units are properly identified and extracted. If your existing models are already of one of the dimensions defined in ChemDataExtractor, i.e. Length, Mass, Time, or Temperature, then it's easy. Just remove value and units properties, as those are included by default, and write the model as a subclass of the appropriate model.

For example, the `BoilingPoint` class we wrote earlier can be further transformed:

```
from chemdataextractor.model import TemperatureModel, StringType, ModelType
from chemdataextractor.model import Compound

class BoilingPoint(TemperatureModel):
    compound = ModelType(Compound)
    parsers = [BpParser()]
```

Defining your own dimensions is also easy; an example of how it's done within ChemDataExtractor for temperatures is provided below, and further information can be found in the *API documentation*, and in the documentation on *creating new units and dimensions*.

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

import logging

from .quantity_model import QuantityModel
from .unit import Unit
from .dimension import Dimension
from ...parse.elements import W, I, R, Optional, Any, OneOrMore, Not, ZeroOrMore
from ...parse.actions import merge, join

log = logging.getLogger(__name__)


class Temperature(Dimension):
    """
    Dimension subclass for temperatures.
    """
    pass


class TemperatureModel(QuantityModel):
    """
    Model for temperatures.
    """
    dimensions = Temperature()


class TemperatureUnit(Unit):
    """
    Base class for units with dimensions of temperature.
    The standard value for temperature is defined to be a Kelvin, implemented in the␣
↪Kelvin class.
    """
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, magnitude=0.0, powers=None):
        super(TemperatureUnit, self).__init__(Temperature(), magnitude, powers)


class Kelvin(TemperatureUnit):
    """
    Class for Kelvins.
    """

    def convert_value_to_standard(self, value):
        return value

    def convert_value_from_standard(self, value):
        return value

    def convert_error_to_standard(self, error):
        return error

    def convert_error_from_standard(self, error):
        return error


class Celsius(TemperatureUnit):
    """
    Class for Celsius
    """

    def convert_value_to_standard(self, value):
        return value + 273.15

    def convert_value_from_standard(self, value):
        return value - 273.15

    def convert_error_to_standard(self, error):
        return error

    def convert_error_from_standard(self, error):
        return error


class Fahrenheit(TemperatureUnit):
    """
    Class for Fahrenheit.
    """

    def convert_value_to_standard(self, value):
        return (value + 459.67) * (5. / 9.)

    def convert_value_from_standard(self, value):
        return value * (9. / 5.) - 459.67

    def convert_error_to_standard(self, error):
        return error * (5. / 9.)

    def convert_error_from_standard(self, error):
        return error * (9. / 5.)
```

(continues on next page)

(continued from previous page)

```
units_dict = {R('°?(((K|k)elvin(s)?)|K)\.?', group=0): Kelvin,
              R('(°C|((C|c)elsius))\.?', group=0): Celsius,
              R('°?((F|f)ahrenheit|F)\.?', group=0): Fahrenheit,
              R('°|C', group=0): None}
# The final element in units_dict is given to ensure that '°C' is parsed correctly,
# as the tokenizer splits it into two. When a parser element is assigned to None,
# this means that this element will be ignored when extracting units, but will
# be taken into account for autoparsers to extract from sentences.
Temperature.units_dict = units_dict
```

4.4.2 Upgrading Parsers

To define this model is great, but we also need to upgrade the parser to make sure that these properties are actually extracted. Let's continue with the boiling point example to see how we'd change `BpParser` to make it extract this information.

```
import re
from chemdataextractor.parse import R, I, W, Optional, merge
from chemdataextractor.parse.base import BaseSentenceParser
from chemdataextractor.utils import first
from chemdataextractor.model import Compound

prefix = (R(u'^b\.?p\.?$', re.I) | I(u'boiling') + I(u'point')).hide()
units = (W(u'°') + Optional(R(u'^[CFK]\.?$')))(u'units').add_action(merge)
value = R(u'^\d+(\.\d+)?$')(u'value')
bp = (prefix + value + units)(u'bp')

class BpParser(BaseParser):
    root = bp

    def interpret(self, result, start, end):
        try:
            raw_value = first(result.xpath('./value/text()'))
            raw_units = first(result.xpath('./units/text()'))
            boiling_point = self.model(raw_value=raw_value,
                        raw_units=raw_units,
                        value=self.extract_value(raw_value),
                        error=self.extract_error(raw_value),
                        units=self.extract_units(raw_units, strict=True))
            yield boiling_point
        except TypeError as e:
            log.debug(e)
```

These parsers can also be made faster by setting the optional *`trigger_phrase`* attribute. The parse element contained in this attribute is run before the root phrase is run, which can result in substantial performance improvements if the root phrase is large and complicated. However, in the case of `BpParser` above, the root phrase itself is so simple that setting this attribute could make the parser slightly slower. You should consider setting the *`trigger_phrase`* for real, more complicated parsers if you are finding the parser to be running too slowly.

4.4.3 Using Automatic Parsers

This is actually the easiest part of upgrading to take advantage of 2.0's features; you only need to add a basic specifier and not set your own parsers, then ChemDataExtractor will handle it all for you.

```
from chemdataextractor.model import TemperatureModel, StringType, ModelType
from chemdataextractor.model import Compound
from chemdataextractor.parse.actions import join
from chemdataextractor.parse import I

class BoilingPoint(TemperatureModel):
    specifier = StringType(parse_expression=(I('Boiling') + I('Point')).add_
↪action(join), required=True)
    compound = ModelType(Compound)
```

Alternatively, if you want to use the parser you wrote yourself instead of the automatic sentence parser, you can do the following:

```
from chemdataextractor.model import TemperatureModel, StringType, ModelType
from chemdataextractor.model import Compound
   from chemdataextractor.parse.actions import join
from chemdataextractor.parse import I
from chemdataextractor.parse.auto import AutoSentenceParser, AutoTableParser


class BoilingPoint(TemperatureModel):
    specifier = StringType(parse_expression=(I('Boiling') + I('Point')).add_
↪action(join), required=True)
    compound = ModelType(Compound)
    parsers = [BpParser(), AutoTableParser()]
```

Note: All parsers added to a class under parsers will be run on the document, so it's best not to have more than one parser which acts on the same type of element to avoid having a large number of duplicated results.

Note: For autoparsers to work correctly, it is **strongly** recommended that you set `required=True` on specifier, but in that case, it's also important that you set some value for the specifier (it doesn't matter what) when extracted with a manual parser, else the record will not be returned.

Also key to making autoparsers work correctly is to always include `add_action(join)` to the end of any parse expressions to ensure that multi-word parse expressions can be picked up correctly by the autoparser.

4.4.4 Fully Nested Models

v2.0 brings the capability to nest models within other models. A simple example of this is that many models, such as the `BoilingPoint` model we defined earlier, contains a model for compound. However, this also works with user-defined properties, and each of these models only needs to parse its surface-level properties, with everything else being merged in later. This nesting can in theory go multiple levels.

As a toy example, say we wanted to associate some additional properties to the boiling point, like the specific heat capacity of the material, and we're in turn interested in the dimensions of the apparatus used to measure the specific heat capacity:

```
from chemdataextractor.model import TemperatureModel, LengthModel, StringType,␣
↪ModelType, QuantityModel, Compound
from chemdataextractor.model.units import Length, Mass, Temperature, Time
from chemdataextractor.parse.actions import join
from chemdataextractor.parse import I
```

(continues on next page)

(continued from previous page)

```
from chemdataextractor.doc import Document, Paragraph, Heading


class ApparatusLength(LengthModel):
    specifier = StringType(parse_expression=(I('measured') + I('with')).add_
↪action(join), required=True)


class SpecificHeatCapacity(QuantityModel):
    dimensions = ((Length() ** 2) * Mass()) / ((Time() ** 2) * Temperature())
    specifier = StringType(parse_expression=(I('Specific') + I('Heat') + I('Capacity
↪')).add_action(join), required=True)
    apparatuslength = ModelType(ApparatusLength, contextual=True)


class BoilingPoint(TemperatureModel):
    specifier = StringType(parse_expression=(I('Boiling') + I('Point')).add_
↪action(join), required=True)
    compound = ModelType(Compound, contextual=True)
    heat_capacity = ModelType(SpecificHeatCapacity, required=True, contextual=True)


document = Document(
    Heading('H2O boiling point, measured with a 200cm long apparatus'),
    Paragraph('H2O was found to have a boiling point of 100 °C, with a specific heat␣
↪capacity of 200 kgm2K-1s-2).'))
document.models = [BoilingPoint]
print(document.records.serialize())
```

The above code will print:

```
 [{'BoilingPoint':
{'raw_value': '100',
'raw_units': '°C',
'value': [100.0],
'units': 'Celsius^(1.0)',
'specifier': 'boiling point',
'compound': {'Compound': {'names': ['H2O']}},
'heat_capacity': {'SpecificHeatCapacity':
                  {'raw_value': '200',
                  'raw_units': 'kgm2K-1s-2',
                  'value': [200.0],
                  'units': '(10^3.0) * Gram^(1.0)  Kelvin^(-1.0)  Meter^(2.0)  Second^
↪(-2.0)',
                  'specifier': 'specific heat capacity',
                  'apparatuslength': {'ApparatusLength':
                                      {'raw_value': '200',
                                      'raw_units': 'cm',
                                      'value': [200.0],
                                      'units': '(10^-2.0) * Meter^(1.0)',
                                      'specifier': 'measured with'}}}}}}]
```

Tip: Complex hierarchies of nested models are now possible, taking the appropriate `required` flags into account for each nested model. For table data, models will automatically be extracted and merged appropriately, regardless of the `contextual` flags, as long as all the submodels share a common `compound` element.

CHAPTER 5

Change Log

5.1 v2.0 (2019-09-xx)

Full Changelog

Implemented enhancements:

- New model structure changed so that the Compound class is no longer at the root of all properties
- Hierarchy changed so that documents own models, not parsers, so that the user doesn't need to remember to pass in all the correct parsers.
- Quantity based models, allowing for easy detection of units and values. Also allows for better comparisons of models.
- Completely new table parsing routine with the incorporation of TableDataExtractor. This returns a more structured form for tables without any user input.
- Automatically generated parsers based on the dimensional information of properties.
- Forward looking Interdependency Resolution for detecting definitions of specifier terms and chemical names.
- Improved Interdependency Resolution to account for more complex models.
- Snowball integration where Snowball parsers can be used seamlessly alongside rule-based parsers.
- Improved performance, with parsing up to 2x faster in real-world usage.
- The incorporation of an evaluation package for measuring the performance of CDE.
- Improved tokenisation when using new quantity based models.
- Improved documentation, including a migration guide for users coming from older versions.

5.2 v1.3.0 (2017-02-03)

Full Changelog

Implemented enhancements:

- Add parser for glass transition temperature #13 (rtchoua)

5.3 v1.2.3 (2017-01-22)

Full Changelog

Fixed bugs:

- _in_stoplist should return True for entities trimmed out of existence #12

5.4 v1.2.2 (2016-11-02)

Full Changelog

Fixed bugs:

- Fix issues with reference link extraction using HTML/XML readers #10 (mcs07)

5.5 v1.2.1 (2016-10-24)

Full Changelog

Fixed bugs:

- RSCHTMLReader throws bytes/string error #8
- Fix encoding bug in RSC image character handling #9 (mcs07)

5.6 v1.2.0 (2016-10-11)

Full Changelog

Implemented enhancements:

- New model layer #5 (mcs07)

Fixed bugs:

- import error: HTMLParser in Python 3 #7
- Installation on Windows 7 #3
- HTML unescape py2/3 compat - fixes #4 #6 (mcs07)

5.7 v1.1.1 (2016-10-04)

Full Changelog

Implemented enhancements:

- Python 3 compatibility #2 (mcs07)

Fixed bugs:

- version of pdfminer #1

5.8 v1.1.0 (2016-10-03)

- *This Change Log was automatically generated by github_changelog_generator*

CHAPTER 6

License/Citing

If you use ChemDataExtractor v2 and/or TableDataExtractor as a resource in your research, please cite the following work:

> Juraj Mavračić, Callum J. Court, Taketomo Isazawa, Stephen R. Elliott, Jacqueline M. Cole: ChemDataExtractor 2.0: Auto-Populated Ontologies for Materials Science", J. Chem. Inf. Model. **2020**, *xx* (xx), pp xxxx–xxxx 10.1021/acs.jcim.xxxxxxx

ChemDataExtractor v2 is based on ChemDataExtractor. So, please also cite the following where relevant:

> Swain, M. C., & Cole, J. M. "ChemDataExtractor: A Toolkit for Automated Extraction of Chemical Information from the Scientific Literature", J. Chem. Inf. Model. **2016**, *56* (10), pp 1894–1904 10.1021/acs.jcim.6b00207

This project was financially supported by the Engineering and Physical Sciences Research Council (EPSRC, EP/L015552/1), Science and Technology Facilities Council (STFC), the Royal Academy of Engineering (RCSRF1819710), and BASF.

6.1 ChemDataExtractor v2 is released under the MIT license.

The MIT License

Copyright 2020 Juraj Mavračić, Callum J. Court, Taketomo Isazawa, Jacqueline M. Cole and contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE

6.2 ChemDataExtractor License

The MIT License

Copyright 2017 Matt Swain and contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

chemdataextractor, 29
chemdataextractor.biblio, 31
chemdataextractor.biblio.bibtex, 31
chemdataextractor.biblio.person, 32
chemdataextractor.biblio.xmp, 33
chemdataextractor.cli, 34
chemdataextractor.cli.cem, 34
chemdataextractor.cli.chemdner, 34
chemdataextractor.cli.cluster, 35
chemdataextractor.cli.config, 35
chemdataextractor.cli.data, 35
chemdataextractor.cli.dict, 35
chemdataextractor.cli.evaluate, 36
chemdataextractor.cli.pos, 37
chemdataextractor.cli.tokenize, 37
chemdataextractor.config, 29
chemdataextractor.data, 30
chemdataextractor.doc, 37
chemdataextractor.doc.document, 37
chemdataextractor.doc.element, 40
chemdataextractor.doc.figure, 42
chemdataextractor.doc.meta, 43
chemdataextractor.doc.table, 44
chemdataextractor.doc.text, 45
chemdataextractor.errors, 30
chemdataextractor.eval, 56
chemdataextractor.model, 56
chemdataextractor.model.base, 57
chemdataextractor.model.model, 63
chemdataextractor.model.units, 68
chemdataextractor.model.units.dimension, 70
chemdataextractor.model.units.length, 73
chemdataextractor.model.units.mass, 75
chemdataextractor.model.units.quantity_model, 71
chemdataextractor.model.units.temperature, 78
chemdataextractor.model.units.time, 76
chemdataextractor.model.units.unit, 68
chemdataextractor.nlp, 79
chemdataextractor.nlp.abbrev, 79
chemdataextractor.nlp.cem, 80
chemdataextractor.nlp.corpus, 81
chemdataextractor.nlp.lexicon, 82
chemdataextractor.nlp.pos, 85
chemdataextractor.nlp.tag, 85
chemdataextractor.nlp.tokenize, 88
chemdataextractor.parse, 91
chemdataextractor.parse.actions, 91
chemdataextractor.parse.auto, 91
chemdataextractor.parse.base, 93
chemdataextractor.parse.cem, 94
chemdataextractor.parse.common, 96
chemdataextractor.parse.context, 96
chemdataextractor.parse.elements, 96
chemdataextractor.parse.ir, 100
chemdataextractor.parse.mp, 100
chemdataextractor.parse.nmr, 101
chemdataextractor.parse.template, 101
chemdataextractor.parse.tg, 103
chemdataextractor.parse.uvvis, 103
chemdataextractor.reader, 104
chemdataextractor.reader.acs, 104
chemdataextractor.reader.base, 104
chemdataextractor.reader.cssp, 105
chemdataextractor.reader.elsevier, 109
chemdataextractor.reader.markup, 105
chemdataextractor.reader.nlm, 106
chemdataextractor.reader.pdf, 107
chemdataextractor.reader.plaintext, 107
chemdataextractor.reader.rsc, 108
chemdataextractor.reader.springer, 110
chemdataextractor.reader.uspto, 108
chemdataextractor.relex.cluster, 111
chemdataextractor.relex.entity, 112
chemdataextractor.relex.pattern, 113

chemdataextractor.relex.phrase, 113
chemdataextractor.relex.relationship, 113
chemdataextractor.relex.utils, 114
chemdataextractor.scrape, 115
chemdataextractor.scrape.base, 129
chemdataextractor.scrape.clean, 131
chemdataextractor.scrape.csstranslator, 132
chemdataextractor.scrape.entity, 133
chemdataextractor.scrape.fields, 135
chemdataextractor.scrape.pub, 115
chemdataextractor.scrape.pub.elsevier, 125
chemdataextractor.scrape.pub.nlm, 115
chemdataextractor.scrape.pub.rsc, 118
chemdataextractor.scrape.pub.springer, 122
chemdataextractor.scrape.scraper, 137
chemdataextractor.scrape.selector, 139
chemdataextractor.text, 140
chemdataextractor.text.chem, 141
chemdataextractor.text.latex, 142
chemdataextractor.text.normalize, 142
chemdataextractor.text.processors, 143
chemdataextractor.utils, 31

Index

Symbols

__init__() (*chemdataextractor.biblio.bibtex.BibtexParser method*), 31
__init__() (*chemdataextractor.biblio.person.PersonName method*), 33
__init__() (*chemdataextractor.biblio.xmp.XmpParser method*), 33
__init__() (*chemdataextractor.config.Config method*), 30
__init__() (*chemdataextractor.data.Package method*), 30
__init__() (*chemdataextractor.doc.document.Document method*), 38
__init__() (*chemdataextractor.doc.element.BaseElement method*), 41
__init__() (*chemdataextractor.doc.element.CaptionedElement method*), 41
__init__() (*chemdataextractor.doc.figure.Figure method*), 42
__init__() (*chemdataextractor.doc.meta.MetaData method*), 43
__init__() (*chemdataextractor.doc.table.Table method*), 44
__init__() (*chemdataextractor.doc.text.BaseText method*), 45
__init__() (*chemdataextractor.doc.text.Caption method*), 51
__init__() (*chemdataextractor.doc.text.Cell method*), 54
__init__() (*chemdataextractor.doc.text.Footnote method*), 51
__init__() (*chemdataextractor.doc.text.Heading method*), 49
__init__() (*chemdataextractor.doc.text.Paragraph method*), 50
__init__() (*chemdataextractor.doc.text.Sentence method*), 53
__init__() (*chemdataextractor.doc.text.Span method*), 55
__init__() (*chemdataextractor.doc.text.Text method*), 46
__init__() (*chemdataextractor.doc.text.Title method*), 48
__init__() (*chemdataextractor.doc.text.Token method*), 56
__init__() (*chemdataextractor.model.base.BaseModel method*), 59
__init__() (*chemdataextractor.model.base.BaseType method*), 57
__init__() (*chemdataextractor.model.base.ListType method*), 58
__init__() (*chemdataextractor.model.base.ModelList method*), 62
__init__() (*chemdataextractor.model.base.ModelType method*), 58
__init__() (*chemdataextractor.model.base.SetType method*), 59
__init__() (*chemdataextractor.model.units.length.LengthUnit method*), 74
__init__() (*chemdataextractor.model.units.mass.MassUnit method*), 75
__init__() (*chemdataextractor.model.units.temperature.TemperatureUnit method*), 78
__init__() (*chemdataextractor.model.units.time.TimeUnit method*), 77
__init__() (*chemdataextractor.model.units.unit.DimensionlessUnit method*), 70
__init__() (*chemdataextractor.model.units.unit.Unit method*), 69
__init__() (*chemdataextractor.nlp.abbrev.AbbreviationDetector method*), 79
__init__() (*chemdataextractor.nlp.corpus.LazyCorpusLoader method*),

81
__init__() (chemdataextractor.nlp.lexicon.Lexeme method), 82
__init__() (chemdataextractor.nlp.lexicon.Lexicon method), 84
__init__() (chemdataextractor.nlp.tag.ApTagger method), 87
__init__() (chemdataextractor.nlp.tag.AveragedPerceptron method), 86
__init__() (chemdataextractor.nlp.tag.CrfTagger method), 87
__init__() (chemdataextractor.nlp.tag.DictionaryTagger method), 88
__init__() (chemdataextractor.nlp.tag.RegexTagger method), 86
__init__() (chemdataextractor.nlp.tokenize.SentenceTokenizer method), 89
__init__() (chemdataextractor.nlp.tokenize.WordTokenizer method), 90
__init__() (chemdataextractor.parse.auto.AutoSentenceParser method), 92
__init__() (chemdataextractor.parse.auto.AutoTableParser method), 92
__init__() (chemdataextractor.parse.auto.BaseAutoParser method), 92
__init__() (chemdataextractor.parse.elements.And method), 98
__init__() (chemdataextractor.parse.elements.BaseParserElement method), 96
__init__() (chemdataextractor.parse.elements.End method), 98
__init__() (chemdataextractor.parse.elements.First method), 98
__init__() (chemdataextractor.parse.elements.IWord method), 98
__init__() (chemdataextractor.parse.elements.Optional method), 99
__init__() (chemdataextractor.parse.elements.ParseElementEnhance method), 99
__init__() (chemdataextractor.parse.elements.ParseException method), 96
__init__() (chemdataextractor.parse.elements.ParseExpression method), 98
__init__() (chemdataextractor.parse.elements.Regex method), 98
__init__() (chemdataextractor.parse.elements.SkipTo method), 100
__init__() (chemdataextractor.parse.elements.Start method), 98
__init__() (chemdataextractor.parse.elements.Tag method), 97
__init__() (chemdataextractor.parse.elements.Word method), 97
__init__() (chemdataextractor.parse.nmr.NmrParser method), 101
__init__() (chemdataextractor.reader.base.BaseReader method), 104
__init__() (chemdataextractor.relex.cluster.Cluster method), 111
__init__() (chemdataextractor.relex.entity.Entity method), 112
__init__() (chemdataextractor.relex.pattern.Pattern method), 113
__init__() (chemdataextractor.relex.phrase.Phrase method), 113
__init__() (chemdataextractor.relex.relationship.Relation method), 114
__init__() (chemdataextractor.scrape.base.BaseField method), 130
__init__() (chemdataextractor.scrape.base.BaseScraper method), 129
__init__() (chemdataextractor.scrape.clean.Cleaner method), 132
__init__() (chemdataextractor.scrape.entity.Entity method), 133
__init__() (chemdataextractor.scrape.entity.EntityList method), 134
__init__() (chemdataextractor.scrape.fields.BoolField method), 136
__init__() (chemdataextractor.scrape.fields.EntityField method), 136
__init__() (chemdataextractor.scrape.fields.StringField method), 135
__init__() (chemdataextractor.scrape.fields.UrlField method), 135
__init__() (chemdataextractor.scrape.pub.rsc.RscSearchScraper method), 119
__init__() (chemdataextractor.scrape.scraper.ResponseSearchResult method), 138
__init__() (chemdataextractor.scrape.scraper.SeleniumSearchResult method), 138
__init__() (chemdataextractor.scrape.selector.Selector method), 139
__init__() (chemdataextractor.scrape.selector.SelectorList method),

139
__init__() (chemdataextractor.text.normalize.ChemNormalizer method), 143
__init__() (chemdataextractor.text.normalize.ExcessNormalizer method), 143
__init__() (chemdataextractor.text.normalize.Normalizer method), 142
__init__() (chemdataextractor.text.processors.Chain method), 144
__init__() (chemdataextractor.text.processors.Discard method), 144
__init__() (chemdataextractor.text.processors.LAdd method), 144
__init__() (chemdataextractor.text.processors.LStrip method), 144
__init__() (chemdataextractor.text.processors.RAdd method), 144
__init__() (chemdataextractor.text.processors.RStrip method), 144
__init__() (chemdataextractor.text.processors.Substitutor method), 144

A

abbr_equivs (chemdataextractor.nlp.abbrev.AbbreviationDetector attribute), 80
abbr_equivs (chemdataextractor.nlp.abbrev.ChemAbbreviationDetector attribute), 80
abbr_max (chemdataextractor.nlp.abbrev.AbbreviationDetector attribute), 80
abbr_max (chemdataextractor.nlp.abbrev.ChemAbbreviationDetector attribute), 80
abbr_min (chemdataextractor.nlp.abbrev.AbbreviationDetector attribute), 80
abbr_min (chemdataextractor.nlp.abbrev.ChemAbbreviationDetector attribute), 80
abbreviation_definitions (chemdataextractor.doc.document.Document attribute), 40
abbreviation_definitions (chemdataextractor.doc.element.CaptionedElement attribute), 42
abbreviation_definitions (chemdataextractor.doc.meta.MetaData attribute), 44
abbreviation_definitions (chemdataextractor.doc.text.Cell attribute), 55
abbreviation_definitions (chemdataextractor.doc.text.Sentence attribute), 54
abbreviation_definitions (chemdataextractor.doc.text.Text attribute), 48
abbreviation_detector (chemdataextractor.doc.text.Citation attribute), 51
abbreviation_detector (chemdataextractor.doc.text.Sentence attribute), 52
abbreviation_detector (chemdataextractor.doc.text.Text attribute), 46
AbbreviationDetector (class in chemdataextractor.nlp.abbrev), 79
abstract (chemdataextractor.scrape.entity.DocumentEntity attribute), 134
abstract (chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute), 126
abstract (chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute), 128
abstract (chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute), 117
abstract (chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute), 121
abstract (chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute), 119
abstract (chemdataextractor.scrape.pub.springer.SpringerHtmlDocument attribute), 122
abstract (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 124
ACCENTS (in module chemdataextractor.text), 140
accepted_day (chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute), 117
accepted_day (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 124
accepted_month (chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute), 117
accepted_month (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 124
accepted_year (chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute), 117
accepted_year (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 124
AcsHtmlReader (class in chemdataextrac-

tor.reader.acs), 104
actions (*chemdataextractor.parse.elements.BaseParserElement attribute*), 96
add() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
add_action() (*chemdataextractor.parse.elements.BaseParserElement method*), 96
add_models() (*chemdataextractor.doc.document.Document method*), 38
add_models() (*chemdataextractor.doc.element.BaseElement method*), 41
add_phrase() (*chemdataextractor.relex.cluster.Cluster method*), 111
add_tokens() (*chemdataextractor.relex.cluster.Cluster static method*), 112
allow_xpath (*chemdataextractor.scrape.clean.Cleaner attribute*), 131
amino_acid (*in module chemdataextractor.parse.cem*), 95
And (*class in chemdataextractor.parse.elements*), 98
Angstrom (*class in chemdataextractor.model.units.length*), 74
another_label (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
Any (*class in chemdataextractor.parse.elements*), 97
APOSTROPHES (*in module chemdataextractor.text*), 140
apparatus (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
apparatus (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
apparatus (*chemdataextractor.model.model.IrSpectrum attribute*), 64
apparatus (*chemdataextractor.model.model.MeltingPoint attribute*), 65
apparatus (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
apparatus (*chemdataextractor.model.model.QuantumYield attribute*), 66
apparatus (*chemdataextractor.model.model.UvvisSpectrum attribute*), 64
Apparatus (*class in chemdataextractor.model.model*), 63
append() (*chemdataextractor.parse.elements.ParseExpression method*), 98
ApPosTagger (*class in chemdataextractor.nlp.pos*), 85
ApTagger (*class in chemdataextractor.nlp.tag*), 86
assignment (*chemdataextractor.model.model.NmrPeak attribute*), 64
attribute (*chemdataextractor.scrape.csstranslator.CdeXPathExpr attribute*), 133
authors (*chemdataextractor.doc.meta.MetaData attribute*), 43
authors (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
authors (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 126
authors (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
authors (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 116
authors (*chemdataextractor.scrape.pub.rsc.RscRssDocument attribute*), 118
authors (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 123
AutoSentenceParser (*class in chemdataextractor.parse.auto*), 92
AutoTableParser (*class in chemdataextractor.parse.auto*), 92
average_weights() (*chemdataextractor.nlp.tag.AveragedPerceptron method*), 86
AveragedPerceptron (*class in chemdataextractor.nlp.tag*), 86

B

base_magnitude (*chemdataextractor.model.units.unit.Unit attribute*), 69
BaseAutoParser (*class in chemdataextractor.parse.auto*), 92
BaseDocument (*class in chemdataextractor.doc.document*), 37
BaseElement (*class in chemdataextractor.doc.element*), 40
BaseEntity (*class in chemdataextractor.scrape.base*), 130
BaseEntityProcessor (*class in chemdataextractor.scrape.base*), 130
BaseField (*class in chemdataextractor.scrape.base*), 130
BaseFormat (*class in chemdataextractor.scrape.base*), 130

BaseModel (*class in chemdataextractor.model.base*), 59
BaseNormalizer (*class in chemdataextractor.text.normalize*), 142
BaseParser (*class in chemdataextractor.parse.base*), 93
BaseParserElement (*class in chemdataextractor.parse.elements*), 96
BaseProcessor (*class in chemdataextractor.text.processors*), 143
BaseReader (*class in chemdataextractor.reader.base*), 104
BaseRequester (*class in chemdataextractor.scrape.base*), 130
BaseScraper (*class in chemdataextractor.scrape.base*), 129
BaseSentenceParser (*class in chemdataextractor.parse.base*), 94
BaseTableParser (*class in chemdataextractor.parse.base*), 94
BaseTagger (*class in chemdataextractor.nlp.tag*), 85
BaseText (*class in chemdataextractor.doc.text*), 45
BaseTokenizer (*class in chemdataextractor.nlp.tokenize*), 88
BaseType (*class in chemdataextractor.model.base*), 57
BibtexParser (*class in chemdataextractor.biblio.bibtex*), 31
binding_properties (*chemdataextractor.model.base.BaseModel attribute*), 62
BLOCK_ELEMENTS (*in module chemdataextractor.scrape*), 115
bond (*chemdataextractor.model.model.IrPeak attribute*), 64
BoolField (*class in chemdataextractor.scrape.fields*), 136
bracket_level() (*in module chemdataextractor.text*), 141
build() (*chemdataextractor.nlp.tag.DictionaryTagger method*), 88
build() (*in module chemdataextractor.cli.dict*), 36

C

caption (*chemdataextractor.scrape.pub.elsevier.ElsevierImage attribute*), 126
caption (*chemdataextractor.scrape.pub.elsevier.ElsevierTable attribute*), 126
caption (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlImage attribute*), 127
caption (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlTable attribute*), 128
caption (*chemdataextractor.scrape.pub.nlm.NlmXmlImage attribute*), 116
caption (*chemdataextractor.scrape.pub.nlm.NlmXmlTable attribute*), 116
caption (*chemdataextractor.scrape.pub.rsc.RscImage attribute*), 121
caption (*chemdataextractor.scrape.pub.rsc.RscTable attribute*), 121
caption (*chemdataextractor.scrape.pub.springer.SpringerXmlImage attribute*), 123
caption (*chemdataextractor.scrape.pub.springer.SpringerXmlTable attribute*), 123
Caption (*class in chemdataextractor.doc.text*), 51
captioned_elements (*chemdataextractor.doc.document.Document attribute*), 40
CaptionedElement (*class in chemdataextractor.doc.element*), 41
captions (*chemdataextractor.doc.document.Document attribute*), 40
case_sensitive (*chemdataextractor.nlp.cem.CsDictCemTagger attribute*), 81
case_sensitive (*chemdataextractor.nlp.tag.DictionaryTagger attribute*), 88
cde_tokensc (*in module chemdataextractor.nlp.corpus*), 82
CdeXPathExpr (*class in chemdataextractor.scrape.csstranslator*), 133
Cell (*class in chemdataextractor.doc.text*), 54
Celsius (*class in chemdataextractor.model.units.temperature*), 79
cem() (*in module chemdataextractor.cli.cem*), 34
cem_after_specifier_and_value_phrase (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
cem_before_specifier_and_value_phrase (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
cem_phrase (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
cems (*chemdataextractor.doc.document.Document attribute*), 40
cems (*chemdataextractor.doc.element.CaptionedElement attribute*), 42
cems (*chemdataextractor.doc.meta.MetaData attribute*), 44

cems (*chemdataextractor.doc.text.Sentence attribute*), 54
cems (*chemdataextractor.doc.text.Text attribute*), 48
CemTagger (*class in chemdataextractor.nlp.cem*), 81
Chain (*class in chemdataextractor.text.processors*), 144
CHAR_REPLACEMENTS (*in module chemdataextractor.scrape.pub.elsevier*), 125
CHAR_REPLACEMENTS (*in module chemdataextractor.scrape.pub.rsc*), 118
ChemAbbreviationDetector (*class in chemdataextractor.nlp.abbrev*), 80
ChemApPosTagger (*class in chemdataextractor.nlp.pos*), 85
ChemCrfPosTagger (*class in chemdataextractor.nlp.pos*), 85
chemdataextractor (*module*), 29
chemdataextractor.biblio (*module*), 31
chemdataextractor.biblio.bibtex (*module*), 31
chemdataextractor.biblio.person (*module*), 32
chemdataextractor.biblio.xmp (*module*), 33
chemdataextractor.cli (*module*), 34
chemdataextractor.cli.cem (*module*), 34
chemdataextractor.cli.chemdner (*module*), 34
chemdataextractor.cli.cluster (*module*), 35
chemdataextractor.cli.config (*module*), 35
chemdataextractor.cli.data (*module*), 35
chemdataextractor.cli.dict (*module*), 35
chemdataextractor.cli.evaluate (*module*), 36
chemdataextractor.cli.pos (*module*), 37
chemdataextractor.cli.tokenize (*module*), 37
chemdataextractor.config (*module*), 29
chemdataextractor.data (*module*), 30
chemdataextractor.doc (*module*), 37
chemdataextractor.doc.document (*module*), 37
chemdataextractor.doc.element (*module*), 40
chemdataextractor.doc.figure (*module*), 42
chemdataextractor.doc.meta (*module*), 43
chemdataextractor.doc.table (*module*), 44
chemdataextractor.doc.text (*module*), 45
chemdataextractor.errors (*module*), 30
chemdataextractor.eval (*module*), 56
chemdataextractor.model (*module*), 56
chemdataextractor.model.base (*module*), 57
chemdataextractor.model.model (*module*), 63
chemdataextractor.model.units (*module*), 68
chemdataextractor.model.units.dimension (*module*), 70
chemdataextractor.model.units.length (*module*), 73
chemdataextractor.model.units.mass (*module*), 75
chemdataextractor.model.units.quantity_model (*module*), 71
chemdataextractor.model.units.temperature (*module*), 78
chemdataextractor.model.units.time (*module*), 76
chemdataextractor.model.units.unit (*module*), 68
chemdataextractor.nlp (*module*), 79
chemdataextractor.nlp.abbrev (*module*), 79
chemdataextractor.nlp.cem (*module*), 80
chemdataextractor.nlp.corpus (*module*), 81
chemdataextractor.nlp.lexicon (*module*), 82
chemdataextractor.nlp.pos (*module*), 85
chemdataextractor.nlp.tag (*module*), 85
chemdataextractor.nlp.tokenize (*module*), 88
chemdataextractor.parse (*module*), 91
chemdataextractor.parse.actions (*module*), 91
chemdataextractor.parse.auto (*module*), 91
chemdataextractor.parse.base (*module*), 93
chemdataextractor.parse.cem (*module*), 94
chemdataextractor.parse.common (*module*), 96
chemdataextractor.parse.context (*module*), 96
chemdataextractor.parse.elements (*module*), 96
chemdataextractor.parse.ir (*module*), 100
chemdataextractor.parse.mp (*module*), 100
chemdataextractor.parse.nmr (*module*), 101
chemdataextractor.parse.template (*module*), 101
chemdataextractor.parse.tg (*module*), 103
chemdataextractor.parse.uvvis (*module*), 103
chemdataextractor.reader (*module*), 104
chemdataextractor.reader.acs (*module*), 104
chemdataextractor.reader.base (*module*), 104
chemdataextractor.reader.cssp (*module*), 105
chemdataextractor.reader.elsevier (*module*), 109
chemdataextractor.reader.markup (*module*), 105
chemdataextractor.reader.nlm (*module*), 106
chemdataextractor.reader.pdf (*module*), 107
chemdataextractor.reader.plaintext (*module*), 107
chemdataextractor.reader.rsc (*module*), 108

chemdataextractor.reader.springer (*module*), 110
chemdataextractor.reader.uspto (*module*), 108
chemdataextractor.relex.cluster (*module*), 111
chemdataextractor.relex.entity (*module*), 112
chemdataextractor.relex.pattern (*module*), 113
chemdataextractor.relex.phrase (*module*), 113
chemdataextractor.relex.relationship (*module*), 113
chemdataextractor.relex.utils (*module*), 114
chemdataextractor.scrape (*module*), 115
chemdataextractor.scrape.base (*module*), 129
chemdataextractor.scrape.clean (*module*), 131
chemdataextractor.scrape.csstranslator (*module*), 132
chemdataextractor.scrape.entity (*module*), 133
chemdataextractor.scrape.fields (*module*), 135
chemdataextractor.scrape.pub (*module*), 115
chemdataextractor.scrape.pub.elsevier (*module*), 125
chemdataextractor.scrape.pub.nlm (*module*), 115
chemdataextractor.scrape.pub.rsc (*module*), 118
chemdataextractor.scrape.pub.springer (*module*), 122
chemdataextractor.scrape.scraper (*module*), 137
chemdataextractor.scrape.selector (*module*), 139
chemdataextractor.text (*module*), 140
chemdataextractor.text.chem (*module*), 141
chemdataextractor.text.latex (*module*), 142
chemdataextractor.text.normalize (*module*), 142
chemdataextractor.text.processors (*module*), 143
chemdataextractor.utils (*module*), 31
ChemDataExtractorError, 30
chemdner_cli() (*in module chemdataextractor.cli.chemdner*), 34
chemdner_training (*in module chemdataextractor.nlp.corpus*), 82
chemical_definitions (*chemdataextractor.doc.element.CaptionedElement attribute*), 42
chemical_definitions (*chemdataextractor.doc.meta.MetaData attribute*), 44
chemical_definitions (*chemdataextractor.doc.text.BaseText attribute*), 46
chemical_definitions (*chemdataextractor.doc.text.Sentence attribute*), 54
chemical_definitions (*chemdataextractor.doc.text.Text attribute*), 48
chemical_label_phrase1 (*in module chemdataextractor.parse.cem*), 95
chemical_label_phrase2 (*in module chemdataextractor.parse.cem*), 95
ChemicalLabelParser (*class in chemdataextractor.parse.cem*), 95
ChemLexicon (*class in chemdataextractor.nlp.lexicon*), 84
ChemNormalizer (*class in chemdataextractor.text.normalize*), 143
ChemSentenceTokenizer (*class in chemdataextractor.nlp.tokenize*), 89
chemspider_id (*chemdataextractor.scrape.pub.rsc.RscChemicalMention attribute*), 120
ChemWordTokenizer (*class in chemdataextractor.nlp.tokenize*), 90
CiDictCemTagger (*class in chemdataextractor.nlp.cem*), 81
Citation (*class in chemdataextractor.doc.text*), 51
citation_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
citation_css (*chemdataextractor.reader.cssp.CsspHtmlReader attribute*), 105
citation_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
citation_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
citation_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
citation_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
citation_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
citations (*chemdataextractor.doc.document.Document attribute*), 39
citations (*chemdataextrac-*

tor.scrape.pub.elsevier.ElsevierHtmlDocument attribute), 127
citations (chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute), 129
clean (in module chemdataextractor.scrape.clean), 132
clean() (in module chemdataextractor.cli.data), 35
clean_abstract (chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute), 118
clean_abstract (chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute), 121
clean_abstract (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 125
clean_acs_html (in module chemdataextractor.reader.acs), 104
clean_caption (chemdataextractor.scrape.pub.nlm.NlmXmlImage attribute), 116
clean_caption (chemdataextractor.scrape.pub.nlm.NlmXmlTable attribute), 116
clean_caption (chemdataextractor.scrape.pub.rsc.RscImage attribute), 121
clean_caption (chemdataextractor.scrape.pub.rsc.RscTable attribute), 121
clean_caption (chemdataextractor.scrape.pub.springer.SpringerXmlImage attribute), 123
clean_caption (chemdataextractor.scrape.pub.springer.SpringerXmlTable attribute), 123
clean_headings (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 125
clean_html (in module chemdataextractor.scrape.clean), 132
clean_html() (chemdataextractor.scrape.clean.Cleaner method), 132
clean_markup (in module chemdataextractor.scrape.clean), 132
clean_markup() (chemdataextractor.scrape.clean.Cleaner method), 132
clean_paragraphs (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 125
clean_src (chemdataextractor.scrape.pub.rsc.RscTable attribute), 121
clean_text (chemdataextractor.scrape.pub.rsc.RscChemicalMention attribute), 120
clean_title (chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute), 118
clean_title (chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute), 121
clean_title (chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute), 119
clean_title (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 125
Cleaner (class in chemdataextractor.scrape.clean), 131
cleaners (chemdataextractor.reader.acs.AcsHtmlReader attribute), 104
cleaners (chemdataextractor.reader.elsevier.ElsevierXmlReader attribute), 109
cleaners (chemdataextractor.reader.markup.LxmlReader attribute), 105
cleaners (chemdataextractor.reader.nlm.NlmXmlReader attribute), 107
cleaners (chemdataextractor.reader.rsc.RscHtmlReader attribute), 108
cleaners (chemdataextractor.reader.springer.SpringerHtmlReader attribute), 111
cleaners (chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute), 110
cleaners (chemdataextractor.reader.uspto.UsptoXmlReader attribute), 108
clear() (chemdataextractor.config.Config method), 30
clear() (in module chemdataextractor.cli.config), 35
cli() (in module chemdataextractor.cli), 34
cluster (chemdataextractor.nlp.lexicon.Lexeme attribute), 83
Cluster (class in chemdataextractor.relex.cluster), 111
cluster() (chemdataextractor.nlp.lexicon.Lexicon method), 84
cluster_cli() (in module chemdataextractor.cli.cluster), 35
clusters (chemdataextractor.nlp.cem.CrfCemTagger attribute), 81
clusters (chemdataextractor.nlp.pos.ApPosTagger attribute), 85
clusters (chemdataextractor.nlp.pos.ChemApPosTagger attribute), 85
clusters (chemdataextrac-

tor.nlp.pos.ChemCrfPosTagger attribute), 85
clusters (*chemdataextractor.nlp.pos.CrfPosTagger attribute*), 85
clusters (*chemdataextractor.nlp.tag.ApTagger attribute*), 87
clusters (*chemdataextractor.nlp.tag.CrfTagger attribute*), 87
clusters_path (*chemdataextractor.nlp.lexicon.ChemLexicon attribute*), 84
clusters_path (*chemdataextractor.nlp.lexicon.Lexicon attribute*), 84
cn_label (*chemdataextractor.model.model.CoordinationNumber attribute*), 68
CNLabel (*class in chemdataextractor.model.model*), 68
coden (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
column_headings (*chemdataextractor.scrape.pub.elsevier.ElsevierTable attribute*), 126
column_headings (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlTable attribute*), 128
compare() (*in module chemdataextractor.cli.evaluate*), 36
compound (*chemdataextractor.model.model.CNLabel attribute*), 68
compound (*chemdataextractor.model.model.CoordinationNumber attribute*), 68
compound (*chemdataextractor.model.model.CurieTemperature attribute*), 67
compound (*chemdataextractor.model.model.GlassTransition attribute*), 65
compound (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
compound (*chemdataextractor.model.model.IrSpectrum attribute*), 64
compound (*chemdataextractor.model.model.MeltingPoint attribute*), 65
compound (*chemdataextractor.model.model.NeelTemperature attribute*), 67
compound (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
compound (*chemdataextractor.model.model.UvvisSpectrum attribute*), 64
Compound (*class in chemdataextractor.model.model*), 63
CompoundHeadingParser (*class in chemdataextractor.parse.cem*), 95
CompoundParser (*class in chemdataextractor.parse.cem*), 95
CompoundTableParser (*class in chemdataextractor.parse.cem*), 95
concentration (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
concentration (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
concentration (*chemdataextractor.model.model.GlassTransition attribute*), 65
concentration (*chemdataextractor.model.model.IrSpectrum attribute*), 64
concentration (*chemdataextractor.model.model.MeltingPoint attribute*), 65
concentration (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
concentration (*chemdataextractor.model.model.QuantumYield attribute*), 66
concentration (*chemdataextractor.model.model.UvvisSpectrum attribute*), 64
concentration_units (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
concentration_units (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
concentration_units (*chemdataextractor.model.model.GlassTransition attribute*), 65
concentration_units (*chemdataextractor.model.model.IrSpectrum attribute*), 64
concentration_units (*chemdataextractor.model.model.MeltingPoint attribute*), 65
concentration_units (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
concentration_units (*chemdataextractor.model.model.QuantumYield attribute*), 66
concentration_units (*chemdataextractor.model.model.UvvisSpectrum attribute*),

64
Config (*class in chemdataextractor.config*), 29
config (*in module chemdataextractor.config*), 30
config_cli() (*in module chemdataextractor.cli.config*), 35
constituent_dimensions (*chemdataextractor.model.units.dimension.Dimension attribute*), 70
constituent_units (*chemdataextractor.model.units.unit.Unit attribute*), 69
construct_category_element() (*in module chemdataextractor.parse.auto*), 92
construct_label_expression() (*chemdataextractor.model.model.Compound method*), 63
construct_unit_element() (*in module chemdataextractor.parse.auto*), 91
construct_yaml_str() (*in module chemdataextractor.config*), 29
contextual_fulfilled (*chemdataextractor.model.base.BaseModel attribute*), 60
CONTRACTIONS (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
CONTROL_RE (*in module chemdataextractor.text*), 141
CONTROLS (*in module chemdataextractor.text*), 140
convert_error() (*chemdataextractor.model.units.quantity_model.QuantityModel method*), 72
convert_error_from_standard() (*chemdataextractor.model.units.length.Angstrom method*), 74
convert_error_from_standard() (*chemdataextractor.model.units.length.LengthUnit method*), 74
convert_error_from_standard() (*chemdataextractor.model.units.length.Meter method*), 74
convert_error_from_standard() (*chemdataextractor.model.units.length.Micron method*), 75
convert_error_from_standard() (*chemdataextractor.model.units.length.Mile method*), 74
convert_error_from_standard() (*chemdataextractor.model.units.mass.Gram method*), 76
convert_error_from_standard() (*chemdataextractor.model.units.mass.MassUnit method*), 75
convert_error_from_standard() (*chemdataextractor.model.units.mass.Pound method*), 76
convert_error_from_standard() (*chemdataextractor.model.units.mass.Tonne method*), 76
convert_error_from_standard() (*chemdataextractor.model.units.temperature.Celsius method*), 79
convert_error_from_standard() (*chemdataextractor.model.units.temperature.Fahrenheit method*), 79
convert_error_from_standard() (*chemdataextractor.model.units.temperature.Kelvin method*), 79
convert_error_from_standard() (*chemdataextractor.model.units.temperature.TemperatureUnit method*), 78
convert_error_from_standard() (*chemdataextractor.model.units.time.Day method*), 78
convert_error_from_standard() (*chemdataextractor.model.units.time.Hour method*), 77
convert_error_from_standard() (*chemdataextractor.model.units.time.Minute method*), 77
convert_error_from_standard() (*chemdataextractor.model.units.time.Second method*), 77
convert_error_from_standard() (*chemdataextractor.model.units.time.TimeUnit method*), 77
convert_error_from_standard() (*chemdataextractor.model.units.time.Year method*), 77
convert_error_from_standard() (*chemdataextractor.model.units.unit.DimensionlessUnit method*), 70
convert_error_from_standard() (*chemdataextractor.model.units.unit.Unit method*), 70
convert_error_to_standard() (*chemdataextractor.model.units.length.Angstrom method*), 74
convert_error_to_standard() (*chemdataextractor.model.units.length.LengthUnit method*), 74
convert_error_to_standard() (*chemdataextractor.model.units.length.Meter method*), 74
convert_error_to_standard() (*chemdataextractor.model.units.length.Micron method*), 75
convert_error_to_standard() (*chemdataextractor.model.units.length.Mile method*), 74

convert_error_to_standard() (*chemdataextractor.model.units.mass.Gram method*), 76
convert_error_to_standard() (*chemdataextractor.model.units.mass.MassUnit method*), 75
convert_error_to_standard() (*chemdataextractor.model.units.mass.Pound method*), 76
convert_error_to_standard() (*chemdataextractor.model.units.mass.Tonne method*), 76
convert_error_to_standard() (*chemdataextractor.model.units.temperature.Celsius method*), 79
convert_error_to_standard() (*chemdataextractor.model.units.temperature.Fahrenheit method*), 79
convert_error_to_standard() (*chemdataextractor.model.units.temperature.Kelvin method*), 79
convert_error_to_standard() (*chemdataextractor.model.units.temperature.TemperatureUnit method*), 78
convert_error_to_standard() (*chemdataextractor.model.units.time.Day method*), 78
convert_error_to_standard() (*chemdataextractor.model.units.time.Hour method*), 77
convert_error_to_standard() (*chemdataextractor.model.units.time.Minute method*), 77
convert_error_to_standard() (*chemdataextractor.model.units.time.Second method*), 77
convert_error_to_standard() (*chemdataextractor.model.units.time.TimeUnit method*), 77
convert_error_to_standard() (*chemdataextractor.model.units.time.Year method*), 77
convert_error_to_standard() (*chemdataextractor.model.units.unit.DimensionlessUnit method*), 70
convert_error_to_standard() (*chemdataextractor.model.units.unit.Unit method*), 70
convert_from_standard() (*chemdataextractor.model.units.time.Year method*), 77
convert_from_standard() (*chemdataextractor.model.units.unit.DimensionlessUnit method*), 70
convert_to() (*chemdataextractor.model.units.quantity_model.QuantityModel method*), 72
convert_to_standard() (*chemdataextractor.model.units.quantity_model.QuantityModel method*), 72
convert_to_standard() (*chemdataextractor.model.units.time.Year method*), 77
convert_to_standard() (*chemdataextractor.model.units.unit.DimensionlessUnit method*), 70
convert_value() (*chemdataextractor.model.units.quantity_model.QuantityModel method*), 72
convert_value_from_standard() (*chemdataextractor.model.units.length.Angstrom method*), 74
convert_value_from_standard() (*chemdataextractor.model.units.length.LengthUnit method*), 74
convert_value_from_standard() (*chemdataextractor.model.units.length.Meter method*), 74
convert_value_from_standard() (*chemdataextractor.model.units.length.Micron method*), 75
convert_value_from_standard() (*chemdataextractor.model.units.length.Mile method*), 74
convert_value_from_standard() (*chemdataextractor.model.units.mass.Gram method*), 76
convert_value_from_standard() (*chemdataextractor.model.units.mass.MassUnit method*), 75
convert_value_from_standard() (*chemdataextractor.model.units.mass.Pound method*), 76
convert_value_from_standard() (*chemdataextractor.model.units.mass.Tonne method*), 76
convert_value_from_standard() (*chemdataextractor.model.units.temperature.Celsius method*), 79
convert_value_from_standard() (*chemdataextractor.model.units.temperature.Fahrenheit method*), 79
convert_value_from_standard() (*chemdataextractor.model.units.temperature.Kelvin method*), 79
convert_value_from_standard() (*chemdataextractor.model.units.temperature.TemperatureUnit method*), 78
convert_value_from_standard() (*chemdataextractor.model.units.time.Day method*), 78
convert_value_from_standard() (*chemdataextractor.model.units.time.Hour method*),

77
convert_value_from_standard() (chemdataextractor.model.units.time.Minute method), 77
convert_value_from_standard() (chemdataextractor.model.units.time.Second method), 77
convert_value_from_standard() (chemdataextractor.model.units.time.TimeUnit method), 77
convert_value_from_standard() (chemdataextractor.model.units.time.Year method), 77
convert_value_from_standard() (chemdataextractor.model.units.unit.DimensionlessUnit method), 70
convert_value_from_standard() (chemdataextractor.model.units.unit.Unit method), 70
convert_value_to_standard() (chemdataextractor.model.units.length.Angstrom method), 74
convert_value_to_standard() (chemdataextractor.model.units.length.LengthUnit method), 74
convert_value_to_standard() (chemdataextractor.model.units.length.Meter method), 74
convert_value_to_standard() (chemdataextractor.model.units.length.Micron method), 75
convert_value_to_standard() (chemdataextractor.model.units.length.Mile method), 74
convert_value_to_standard() (chemdataextractor.model.units.mass.Gram method), 76
convert_value_to_standard() (chemdataextractor.model.units.mass.MassUnit method), 75
convert_value_to_standard() (chemdataextractor.model.units.mass.Pound method), 76
convert_value_to_standard() (chemdataextractor.model.units.mass.Tonne method), 76
convert_value_to_standard() (chemdataextractor.model.units.temperature.Celsius method), 79
convert_value_to_standard() (chemdataextractor.model.units.temperature.Fahrenheit method), 79
convert_value_to_standard() (chemdataextractor.model.units.temperature.Kelvin method), 79
convert_value_to_standard() (chemdataextractor.model.units.temperature.TemperatureUnit method), 79
convert_value_to_standard() (chemdataextractor.model.units.time.Day method), 78
convert_value_to_standard() (chemdataextractor.model.units.time.Hour method), 77
convert_value_to_standard() (chemdataextractor.model.units.time.Minute method), 77
convert_value_to_standard() (chemdataextractor.model.units.time.Second method), 77
convert_value_to_standard() (chemdataextractor.model.units.time.TimeUnit method), 77
convert_value_to_standard() (chemdataextractor.model.units.time.Year method), 77
convert_value_to_standard() (chemdataextractor.model.units.unit.DimensionlessUnit method), 70
convert_value_to_standard() (chemdataextractor.model.units.unit.Unit method), 69
coordination_number_label (chemdataextractor.model.model.CNLabel attribute), 68
coordination_number_label (chemdataextractor.model.model.CoordinationNumber attribute), 67
CoordinationNumber (class in chemdataextractor.model.model), 67
copy() (chemdataextractor.parse.elements.BaseParserElement method), 96
copy() (chemdataextractor.parse.elements.ParseExpression method), 98
copyright (chemdataextractor.scrape.entity.DocumentEntity attribute), 135
copyright (chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute), 127
copyright (chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute), 128
could_be() (chemdataextractor.biblio.person.PersonName method), 33
coupling (chemdataextractor.model.model.NmrPeak attribute), 64
coupling_units (chemdataextractor.model.model.NmrPeak attribute), 64
create() (chemdataextractor.relex.phrase.Phrase method), 113
create_entities_list() (in module chemdataex-

tractor.parse.auto), 92
create_session() (*chemdataextractor.scrape.base.BaseScraper method*), 129
CrfCemTagger (*class in chemdataextractor.nlp.cem*), 81
CrfPosTagger (*class in chemdataextractor.nlp.pos*), 85
CrfTagger (*class in chemdataextractor.nlp.tag*), 87
CsDictCemTagger (*class in chemdataextractor.nlp.cem*), 81
css() (*chemdataextractor.scrape.selector.Selector method*), 139
CssHTMLTranslator (*class in chemdataextractor.scrape.csstranslator*), 133
CsspHtmlReader (*class in chemdataextractor.reader.cssp*), 105
CssXmlTranslator (*class in chemdataextractor.scrape.csstranslator*), 133
CurieTemperature (*class in chemdataextractor.model.model*), 67

D

data (*chemdataextractor.doc.meta.MetaData attribute*), 44
data (*chemdataextractor.scrape.pub.elsevier.ElsevierTable attribute*), 126
data (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlTable attribute*), 128
data_cli() (*in module chemdataextractor.cli.data*), 35
date (*chemdataextractor.doc.meta.MetaData attribute*), 44
DateTimeField (*class in chemdataextractor.scrape.fields*), 136
Day (*class in chemdataextractor.model.units.time*), 78
definitions (*chemdataextractor.doc.document.Document attribute*), 40
definitions (*chemdataextractor.doc.element.CaptionedElement attribute*), 42
definitions (*chemdataextractor.doc.meta.MetaData attribute*), 44
definitions (*chemdataextractor.doc.table.Table attribute*), 45
definitions (*chemdataextractor.doc.text.BaseText attribute*), 46
definitions (*chemdataextractor.doc.text.Caption attribute*), 52
definitions (*chemdataextractor.doc.text.Sentence attribute*), 54
definitions (*chemdataextractor.doc.text.Text attribute*), 48
delimiters (*chemdataextractor.nlp.tag.DictionaryTagger attribute*), 88
detect() (*chemdataextractor.nlp.abbrev.AbbreviationDetector method*), 80
detect() (*chemdataextractor.reader.acs.AcsHtmlReader method*), 104
detect() (*chemdataextractor.reader.base.BaseReader method*), 104
detect() (*chemdataextractor.reader.cssp.CsspHtmlReader method*), 105
detect() (*chemdataextractor.reader.elsevier.ElsevierXmlReader method*), 110
detect() (*chemdataextractor.reader.markup.HtmlReader method*), 106
detect() (*chemdataextractor.reader.markup.XmlReader method*), 106
detect() (*chemdataextractor.reader.nlm.NlmXmlReader method*), 107
detect() (*chemdataextractor.reader.pdf.PdfReader method*), 107
detect() (*chemdataextractor.reader.plaintext.PlainTextReader method*), 107
detect() (*chemdataextractor.reader.rsc.RscHtmlReader method*), 108
detect() (*chemdataextractor.reader.springer.SpringerHtmlReader method*), 111
detect() (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader method*), 110
detect() (*chemdataextractor.reader.uspto.UsptoXmlReader method*), 109
detect_spans() (*chemdataextractor.nlp.abbrev.AbbreviationDetector method*), 80
dict_cli() (*in module chemdataextractor.cli.dict*), 35
DictionaryTagger (*class in chemdataextractor.nlp.tag*), 87
digit_count (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
digit_count() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
Dimension (*class in chemdataextractor.model.units.dimension*), 70
Dimensionless (*class in chemdataextractor.model.units.dimension*), 71
DimensionlessModel (*class in chemdataextractor.model.units.quantity_model*), 73

DimensionlessUnit (*class in chemdataextractor.model.units.unit*), 70
dimensions (*chemdataextractor.model.units.length.LengthModel attribute*), 73
dimensions (*chemdataextractor.model.units.mass.MassModel attribute*), 75
dimensions (*chemdataextractor.model.units.quantity_model.DimensionlessModel attribute*), 73
dimensions (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 72
dimensions (*chemdataextractor.model.units.temperature.TemperatureModel attribute*), 78
dimensions (*chemdataextractor.model.units.time.TimeModel attribute*), 76
Discard (*class in chemdataextractor.text.processors*), 144
document (*chemdataextractor.doc.element.BaseElement attribute*), 41
document (*chemdataextractor.doc.element.CaptionedElement attribute*), 42
Document (*class in chemdataextractor.doc.document*), 38
DocumentEntity (*class in chemdataextractor.scrape.entity*), 134
doi (*chemdataextractor.doc.meta.MetaData attribute*), 43
doi (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
doi (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 126
doi (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
doi (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 116
doi (*chemdataextractor.scrape.pub.rsc.RscRssDocument attribute*), 118
doi (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
doi (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 123
DOI_RE (*in module chemdataextractor.text*), 140
DOUBLE_QUOTES (*in module chemdataextractor.text*), 140
download() (*chemdataextractor.data.Package method*), 30
download() (*in module chemdataextractor.cli.data*), 35

E

ElectrochemicalPotential (*class in chemdataextractor.model.model*), 66
element_symbol (*in module chemdataextractor.parse.cem*), 95
elements (*chemdataextractor.doc.document.BaseDocument attribute*), 38
elements (*chemdataextractor.doc.document.Document attribute*), 39
els_xml_whitespace() (*in module chemdataextractor.reader.elsevier*), 109
elsevier_substitute (*in module chemdataextractor.scrape.pub.elsevier*), 125
ElsevierHtmlDocument (*class in chemdataextractor.scrape.pub.elsevier*), 126
ElsevierHtmlScraper (*class in chemdataextractor.scrape.pub.elsevier*), 127
ElsevierImage (*class in chemdataextractor.scrape.pub.elsevier*), 125
ElsevierSearchDocument (*class in chemdataextractor.scrape.pub.elsevier*), 125
ElsevierSearchScraper (*class in chemdataextractor.scrape.pub.elsevier*), 125
ElsevierTable (*class in chemdataextractor.scrape.pub.elsevier*), 126
ElsevierTableData (*class in chemdataextractor.scrape.pub.elsevier*), 126
ElsevierXmlDocument (*class in chemdataextractor.scrape.pub.elsevier*), 128
ElsevierXmlImage (*class in chemdataextractor.scrape.pub.elsevier*), 127
ElsevierXmlReader (*class in chemdataextractor.reader.elsevier*), 109
ElsevierXmlTable (*class in chemdataextractor.scrape.pub.elsevier*), 127
ElsevierXmlTableData (*class in chemdataextractor.scrape.pub.elsevier*), 127
email (*chemdataextractor.scrape.pub.nlm.NlmXmlAuthor attribute*), 115
email (*chemdataextractor.scrape.pub.springer.SpringerXmlAuthor attribute*), 122
EMAIL_RE (*in module chemdataextractor.text*), 140
end (*chemdataextractor.doc.text.Sentence attribute*), 53
end (*chemdataextractor.doc.text.Span attribute*), 56
End (*class in chemdataextractor.parse.elements*), 98
ensure_dir() (*in module chemdataextractor.utils*), 31
entities (*chemdataextractor.parse.cem.CompoundTableParser attribute*), 95

entity (*chemdataextractor.nlp.tag.DictionaryTagger attribute*), 88
entity (*chemdataextractor.scrape.base.BaseScraper attribute*), 129
entity (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlScraper attribute*), 127
entity (*chemdataextractor.scrape.pub.elsevier.ElsevierSearchScraper attribute*), 125
entity (*chemdataextractor.scrape.pub.rsc.RscHtmlScraper attribute*), 122
entity (*chemdataextractor.scrape.pub.rsc.RscLandingScraper attribute*), 120
entity (*chemdataextractor.scrape.pub.rsc.RscRssScraper attribute*), 119
entity (*chemdataextractor.scrape.pub.rsc.RscSearchScraper attribute*), 119
Entity (*class in chemdataextractor.relex.entity*), 112
Entity (*class in chemdataextractor.scrape.entity*), 133
EntityField (*class in chemdataextractor.scrape.fields*), 136
EntityList (*class in chemdataextractor.scrape.entity*), 134
EntityMeta (*class in chemdataextractor.scrape.base*), 130
error (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 71
eval_document() (*in module chemdataextractor.cli.evaluate*), 36
evaluate() (*chemdataextractor.nlp.tag.BaseTagger method*), 86
evaluate() (*in module chemdataextractor.cli.evaluate*), 36
evaluate() (*in module chemdataextractor.cli.pos*), 37
evaluate_all() (*in module chemdataextractor.cli.pos*), 37
evaluate_perceptron() (*in module chemdataextractor.cli.pos*), 37
ExcessNormalizer (*class in chemdataextractor.text.normalize*), 143
expression (*chemdataextractor.model.model.CurieTemperature attribute*), 67
expression (*chemdataextractor.model.model.NeelTemperature attribute*), 67
extinction (*chemdataextractor.model.model.UvvisPeak attribute*), 63
extinction_units (*chemdataextractor.model.model.UvvisPeak attribute*), 63
extract() (*chemdataextractor.scrape.selector.Selector method*), 139
extract() (*chemdataextractor.scrape.selector.SelectorList method*), 139
extract() (*in module chemdataextractor.cli*), 34
extract_cas() (*in module chemdataextractor.text.chem*), 141
extract_emails() (*in module chemdataextractor.text.processors*), 145
extract_error() (*chemdataextractor.parse.base.BaseParser method*), 93
extract_first() (*chemdataextractor.scrape.selector.SelectorList method*), 139
extract_inchikeys() (*in module chemdataextractor.text.chem*), 141
extract_inchis() (*in module chemdataextractor.text.chem*), 141
extract_smiles() (*in module chemdataextractor.text.chem*), 141
extract_units() (*chemdataextractor.parse.base.BaseParser method*), 93
extract_units() (*in module chemdataextractor.parse.ir*), 100
extract_value() (*chemdataextractor.parse.base.BaseParser method*), 93

F

Fahrenheit (*class in chemdataextractor.model.units.temperature*), 79
fields (*chemdataextractor.model.base.BaseModel attribute*), 59
fields (*chemdataextractor.model.model.Apparatus attribute*), 63
fields (*chemdataextractor.model.model.CNLabel attribute*), 68
fields (*chemdataextractor.model.model.Compound attribute*), 63
fields (*chemdataextractor.model.model.CoordinationNumber attribute*), 68
fields (*chemdataextractor.model.model.CurieTemperature attribute*), 67
fields (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
fields (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66

fields (*chemdataextractor.model.model.GlassTransition attribute*), 66
fields (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
fields (*chemdataextractor.model.model.IrPeak attribute*), 64
fields (*chemdataextractor.model.model.IrSpectrum attribute*), 64
fields (*chemdataextractor.model.model.MeltingPoint attribute*), 65
fields (*chemdataextractor.model.model.NeelTemperature attribute*), 67
fields (*chemdataextractor.model.model.NmrPeak attribute*), 65
fields (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
fields (*chemdataextractor.model.model.QuantumYield attribute*), 66
fields (*chemdataextractor.model.model.UvvisPeak attribute*), 63
fields (*chemdataextractor.model.model.UvvisSpectrum attribute*), 64
fields (*chemdataextractor.model.units.length.LengthModel attribute*), 73
fields (*chemdataextractor.model.units.mass.MassModel attribute*), 75
fields (*chemdataextractor.model.units.quantity_model.DimensionlessModel attribute*), 73
fields (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 73
fields (*chemdataextractor.model.units.temperature.TemperatureModel attribute*), 78
fields (*chemdataextractor.model.units.time.TimeModel attribute*), 76
fields (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
fields (*chemdataextractor.scrape.entity.Entity attribute*), 133
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierImage attribute*), 126
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierSearchDocument attribute*), 125
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierTable attribute*), 126
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierTableData attribute*), 126
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 129
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlImage attribute*), 127
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlTable attribute*), 128
fields (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlTableData attribute*), 127
fields (*chemdataextractor.scrape.pub.nlm.NlmXmlAuthor attribute*), 115
fields (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 118
fields (*chemdataextractor.scrape.pub.nlm.NlmXmlImage attribute*), 116
fields (*chemdataextractor.scrape.pub.nlm.NlmXmlTable attribute*), 116
fields (*chemdataextractor.scrape.pub.rsc.RscChemicalMention attribute*), 120
fields (*chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute*), 122
fields (*chemdataextractor.scrape.pub.rsc.RscImage attribute*), 121
fields (*chemdataextractor.scrape.pub.rsc.RscLandingDocument attribute*), 120
fields (*chemdataextractor.scrape.pub.rsc.RscLandingSupplement attribute*), 120
fields (*chemdataextractor.scrape.pub.rsc.RscRssDocument attribute*), 119
fields (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119

fields (*chemdataextractor.scrape.pub.rsc.RscTable attribute*), 121
fields (*chemdataextractor.scrape.pub.springer.SpringerHtmlDocument attribute*), 122
fields (*chemdataextractor.scrape.pub.springer.SpringerXmlAuthor attribute*), 122
fields (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 125
fields (*chemdataextractor.scrape.pub.springer.SpringerXmlImage attribute*), 123
fields (*chemdataextractor.scrape.pub.springer.SpringerXmlTable attribute*), 123
Figure (*class in chemdataextractor.doc.figure*), 42
figure_caption_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
figure_caption_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
figure_caption_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
figure_caption_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
figure_caption_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
figure_caption_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
figure_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
figure_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
figure_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
figure_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
figure_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
figure_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
figure_download_link_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
figure_download_link_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
figure_download_link_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
figure_label_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
figure_label_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
figure_label_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
figure_label_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
figures (*chemdataextractor.doc.document.Document attribute*), 39
figures (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
figures (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
figures (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
finalize_doi() (*chemdataextractor.scrape.pub.rsc.RscRssDocument method*), 118
find_data() (*in module chemdataextractor.data*), 30
FineWordTokenizer (*class in chemdataextractor.nlp.tokenize*), 90
first (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
First (*class in chemdataextractor.parse.elements*), 98
first() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
first() (*in module chemdataextractor.utils*), 31
firstname (*chemdataextractor.scrape.pub.springer.SpringerXmlAuthor attribute*), 122
firstpage (*chemdataextractor.doc.meta.MetaData attribute*), 43
firstpage (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
firstpage (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128

firstpage (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
firstpage (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 123
fix_elsevier_xml_whitespace() (*in module chemdataextractor.reader.elsevier*), 109
fix_nmr_peak_whitespace_error() (*in module chemdataextractor.parse.nmr*), 101
fix_springer_table_whitespace() (*in module chemdataextractor.reader.springer*), 110
fix_whitespace (*chemdataextractor.scrape.clean.Cleaner attribute*), 132
fix_whitespace() (*in module chemdataextractor.parse.actions*), 91
flatten() (*chemdataextractor.model.base.BaseModel class method*), 61
flatten() (*in module chemdataextractor.parse.actions*), 91
flatten() (*in module chemdataextractor.utils*), 31
FloatField (*class in chemdataextractor.scrape.fields*), 136
floats() (*in module chemdataextractor.text.processors*), 144
FloatType (*class in chemdataextractor.model.base*), 57
FluorescenceLifetime (*class in chemdataextractor.model.model*), 66
FollowedBy (*class in chemdataextractor.parse.elements*), 99
Footnote (*class in chemdataextractor.doc.text*), 50
footnotes (*chemdataextractor.doc.document.Document attribute*), 39
formula (*in module chemdataextractor.parse.cem*), 95
frequency (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
frequency_units (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
from_file() (*chemdataextractor.doc.document.Document class method*), 38
from_html() (*chemdataextractor.scrape.selector.Selector class method*), 139
from_html_text() (*chemdataextractor.scrape.selector.Selector class method*), 139
from_response() (*chemdataextractor.scrape.selector.Selector class method*), 139
from_string() (*chemdataextractor.doc.document.Document class method*), 39
from_tdecell() (*chemdataextractor.doc.text.Cell class method*), 55
from_text() (*chemdataextractor.scrape.selector.Selector class method*), 139
from_xml() (*chemdataextractor.scrape.selector.Selector class method*), 139
from_xml_text() (*chemdataextractor.scrape.selector.Selector class method*), 139
from_xpath() (*chemdataextractor.scrape.csstranslator.CdeXPathExpr class method*), 133
fullname (*chemdataextractor.biblio.person.PersonName attribute*), 33

G

generate_cde_parse_expression() (*chemdataextractor.relex.pattern.Pattern method*), 113
genia_evaluation (*in module chemdataextractor.nlp.corpus*), 82
genia_training (*in module chemdataextractor.nlp.corpus*), 82
get() (*chemdataextractor.model.base.BaseModel method*), 60
get() (*in module chemdataextractor.cli.config*), 35
get_additional_regex() (*chemdataextractor.nlp.tokenize.ChemWordTokenizer method*), 90
get_additional_regex() (*chemdataextractor.nlp.tokenize.WordTokenizer method*), 90
get_data_dir() (*in module chemdataextractor.data*), 30
get_element_with_id() (*chemdataextractor.doc.document.Document method*), 39
get_encoding() (*in module chemdataextractor.text*), 141
get_ids() (*in module chemdataextractor.cli.evaluate*), 36
get_labels() (*in module chemdataextractor.cli.evaluate*), 36
get_names() (*in module chemdataextractor.cli.evaluate*), 36
get_property_apparatus() (*in module chemdataextractor.cli.evaluate*), 36
get_property_core() (*in module chemdataextractor.cli.evaluate*), 36
get_property_full() (*in module chemdataextractor.cli.evaluate*), 36

get_property_solvent() (*in module chemdataextractor.cli.evaluate*), 36
get_property_subject() (*in module chemdataextractor.cli.evaluate*), 36
get_property_temperature() (*in module chemdataextractor.cli.evaluate*), 36
get_property_units() (*in module chemdataextractor.cli.evaluate*), 36
get_property_value() (*in module chemdataextractor.cli.evaluate*), 36
get_relations() (*chemdataextractor.relex.cluster.Cluster method*), 112
get_roots() (*chemdataextractor.scrape.base.BaseScraper method*), 129
get_sentences() (*chemdataextractor.nlp.tokenize.SentenceTokenizer method*), 89
get_spectra_apparatus() (*in module chemdataextractor.cli.evaluate*), 36
get_spectra_core() (*in module chemdataextractor.cli.evaluate*), 36
get_spectra_full() (*in module chemdataextractor.cli.evaluate*), 36
get_spectra_peaks() (*in module chemdataextractor.cli.evaluate*), 36
get_spectra_solvent() (*in module chemdataextractor.cli.evaluate*), 36
get_spectra_subject() (*in module chemdataextractor.cli.evaluate*), 36
get_spectra_temp() (*in module chemdataextractor.cli.evaluate*), 36
get_spectra_type() (*in module chemdataextractor.cli.evaluate*), 36
get_word_tokens() (*chemdataextractor.nlp.tokenize.WordTokenizer method*), 90
GetRequester (*class in chemdataextractor.scrape.scraper*), 137
givennames (*chemdataextractor.scrape.pub.nlm.NlmXmlAuthor attribute*), 115
GlassTransition (*class in chemdataextractor.model.model*), 65
Gram (*class in chemdataextractor.model.units.mass*), 75
GREEK (*in module chemdataextractor.text*), 140
GREEK_WORDS (*in module chemdataextractor.text*), 140
Group (*class in chemdataextractor.parse.elements*), 99

H

H (*in module chemdataextractor.parse.elements*), 100
handle_additional_regex() (*chemdataextractor.nlp.tokenize.WordTokenizer method*), 90
Heading (*class in chemdataextractor.doc.text*), 49
heading_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
heading_css (*chemdataextractor.reader.cssp.CsspHtmlReader attribute*), 105
heading_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
heading_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
heading_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
heading_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
heading_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
heading_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
heading_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 108
headings (*chemdataextractor.doc.document.Document attribute*), 40
headings (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
headings (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
headings (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
Hide (*class in chemdataextractor.parse.elements*), 100
hide() (*chemdataextractor.parse.elements.BaseParserElement method*), 97
hide() (*chemdataextractor.parse.elements.Hide method*), 100
Hour (*class in chemdataextractor.model.units.time*), 77
html_url (*chemdataextractor.doc.meta.MetaData attribute*), 44
html_url (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
html_url (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
html_url (*chemdataextrac-*

tor.scrape.pub.rsc.RscHtmlDocument attribute), 121
html_url (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
HtmlFormat (*class in chemdataextractor.scrape.scraper*), 137
HtmlReader (*class in chemdataextractor.reader.markup*), 106
HYPHENS (*in module chemdataextractor.text*), 140

I

I (*in module chemdataextractor.parse.elements*), 100
ignore_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
ignore_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
ignore_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
ignore_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
ignore_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
ignore_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
ignore_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
ignore_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 109
IGNORE_PREFIX (*in module chemdataextractor.nlp.cem*), 80
IGNORE_SUFFIX (*in module chemdataextractor.nlp.cem*), 80
image_url (*chemdataextractor.scrape.pub.elsevier.ElsevierImage attribute*), 126
inchi (*chemdataextractor.scrape.pub.rsc.RscChemicalMention attribute*), 120
inline_elements (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
inline_elements (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
inline_elements (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 109
INLINE_ELEMENTS (*in module chemdataextractor.scrape*), 115
insert() (*chemdataextractor.model.base.ModelList method*), 62
intensity (*chemdataextractor.model.model.NmrPeak attribute*), 64
InteratomicDistance (*class in chemdataextractor.model.model*), 67
interpret() (*chemdataextractor.parse.auto.BaseAutoParser method*), 92
interpret() (*chemdataextractor.parse.base.BaseParser method*), 93
interpret() (*chemdataextractor.parse.cem.ChemicalLabelParser method*), 95
interpret() (*chemdataextractor.parse.cem.CompoundHeadingParser method*), 95
interpret() (*chemdataextractor.parse.cem.CompoundParser method*), 95
interpret() (*chemdataextractor.parse.cem.CompoundTableParser method*), 95
interpret() (*chemdataextractor.parse.ir.IrParser method*), 100
interpret() (*chemdataextractor.parse.mp.MpParser method*), 100
interpret() (*chemdataextractor.parse.nmr.NmrParser method*), 101
interpret() (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser method*), 103
interpret() (*chemdataextractor.parse.tg.TgParser method*), 103
interpret() (*chemdataextractor.parse.uvvis.UvvisParser method*), 104
interpret_multi_entity_1() (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser method*), 103
interpret_multi_entity_2() (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser method*), 103
interpret_multi_entity_3() (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser method*), 103
interpret_multi_entity_4() (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser method*), 103
IntField (*class in chemdataextractor.scrape.fields*),

136
IrParser (*class in chemdataextractor.parse.ir*), 100
IrPeak (*class in chemdataextractor.model.model*), 64
IrSpectrum (*class in chemdataextractor.model.model*), 64
is_alpha (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_alpha() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_ascii (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_ascii() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_ascii() (*in module chemdataextractor.text*), 141
is_digit (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_digit() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_empty (*chemdataextractor.model.base.BaseModel attribute*), 62
is_empty() (*chemdataextractor.model.base.BaseType method*), 57
is_empty() (*chemdataextractor.model.base.FloatType method*), 58
is_empty() (*chemdataextractor.model.base.ListType method*), 59
is_empty() (*chemdataextractor.model.base.ModelType method*), 58
is_empty() (*chemdataextractor.model.base.SetType method*), 59
is_empty() (*chemdataextractor.model.base.StringType method*), 57
is_empty() (*chemdataextractor.model.units.unit.UnitType method*), 68
is_equal() (*chemdataextractor.model.units.quantity_model.QuantityModel method*), 72
is_hyphenated (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_hyphenated() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_id_only (*chemdataextractor.model.model.Compound attribute*), 63
is_lower (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_lower() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_punct (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_punct() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_punct() (*in module chemdataextractor.text*), 141
is_subset() (*chemdataextractor.model.base.BaseModel method*), 61
is_superset() (*chemdataextractor.model.base.BaseModel method*), 60
is_superset() (*chemdataextractor.model.units.quantity_model.QuantityModel method*), 73
is_title (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_title() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_unidentified (*chemdataextractor.doc.meta.MetaData attribute*), 44
is_unidentified (*chemdataextractor.model.base.BaseModel attribute*), 60
is_unidentified (*chemdataextractor.model.model.Compound attribute*), 63
is_upper (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
is_upper() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
is_valid() (*chemdataextractor.relex.relationship.Relation method*), 114
issn (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
issn (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
issn (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
ISSN_RE (*in module chemdataextractor.text*), 140
issue (*chemdataextractor.doc.meta.MetaData attribute*), 43
issue (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
issue (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
issue (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
issue (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
items() (*chemdataextractor.model.base.BaseModel method*), 60
IWord (*class in chemdataextractor.parse.elements*), 97

J

join() (*chemdataextractor.scrape.csstranslator.CdeXPathExpr method*), 133
join() (*in module chemdataextractor.parse.actions*), 91

join_rsc_table_captions() (*in module chemdataextractor.reader.rsc*), 108
journal (*chemdataextractor.doc.meta.MetaData attribute*), 43
journal (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
journal (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 126
journal (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
journal (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
journal (*chemdataextractor.scrape.pub.springer.SpringerHtmlDocument attribute*), 122
journal (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 123
journal_abbreviation (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 116
journal_title (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 116
json (*chemdataextractor.biblio.bibtex.BibtexParser attribute*), 32

K

Kelvin (*class in chemdataextractor.model.units.temperature*), 79
keys() (*chemdataextractor.model.base.BaseModel method*), 60
kill_xpath (*chemdataextractor.scrape.clean.Cleaner attribute*), 131
KnuthMorrisPratt() (*in module chemdataextractor.relex.utils*), 114

L

label (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlImage attribute*), 127
label (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlTable attribute*), 127
label (*chemdataextractor.scrape.pub.nlm.NlmXmlImage attribute*), 116
label (*chemdataextractor.scrape.pub.nlm.NlmXmlTable attribute*), 116
label (*chemdataextractor.scrape.pub.rsc.RscImage attribute*), 121
label (*chemdataextractor.scrape.pub.rsc.RscTable attribute*), 121
label (*chemdataextractor.scrape.pub.springer.SpringerXmlImage attribute*), 123
label (*chemdataextractor.scrape.pub.springer.SpringerXmlTable attribute*), 123
label_Juraj (*chemdataextractor.model.model.CNLabel attribute*), 68
labels (*chemdataextractor.model.model.Compound attribute*), 63
LAdd (*class in chemdataextractor.text.processors*), 144
landing_url (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
landing_url (*chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute*), 121
landing_url (*chemdataextractor.scrape.pub.rsc.RscRssDocument attribute*), 118
landing_url (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
landing_url (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
language (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
lastname (*chemdataextractor.scrape.pub.nlm.NlmXmlAuthor attribute*), 115
lastname (*chemdataextractor.scrape.pub.springer.SpringerXmlAuthor attribute*), 122
lastpage (*chemdataextractor.doc.meta.MetaData attribute*), 43
lastpage (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
lastpage (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
lastpage (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
latex_to_unicode() (*in module chemdataextractor.text.latex*), 142
LazyCorpusLoader (*class in chemdataextractor.nlp.corpus*), 81

length (*chemdataextractor.doc.text.Span attribute*), 56
length (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
Length (*class in chemdataextractor.model.units.length*), 73
length() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
LengthModel (*class in chemdataextractor.model.units.length*), 73
LengthUnit (*class in chemdataextractor.model.units.length*), 74
levenshtein() (*in module chemdataextractor.text*), 141
lex (*chemdataextractor.doc.text.Token attribute*), 56
Lexeme (*class in chemdataextractor.nlp.lexicon*), 82
lexicon (*chemdataextractor.doc.text.BaseText attribute*), 46
lexicon (*chemdataextractor.doc.text.Sentence attribute*), 52
lexicon (*chemdataextractor.doc.text.Text attribute*), 46
lexicon (*chemdataextractor.doc.text.Token attribute*), 56
lexicon (*chemdataextractor.nlp.cem.CemTagger attribute*), 81
lexicon (*chemdataextractor.nlp.cem.CiDictCemTagger attribute*), 81
lexicon (*chemdataextractor.nlp.cem.CrfCemTagger attribute*), 81
lexicon (*chemdataextractor.nlp.cem.CsDictCemTagger attribute*), 81
lexicon (*chemdataextractor.nlp.pos.ChemApPosTagger attribute*), 85
lexicon (*chemdataextractor.nlp.pos.ChemCrfPosTagger attribute*), 85
lexicon (*chemdataextractor.nlp.tag.ApTagger attribute*), 87
lexicon (*chemdataextractor.nlp.tag.CrfTagger attribute*), 87
lexicon (*chemdataextractor.nlp.tag.DictionaryTagger attribute*), 88
lexicon (*chemdataextractor.nlp.tag.RegexTagger attribute*), 86
Lexicon (*class in chemdataextractor.nlp.lexicon*), 83
license (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
license (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
license (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
like_number (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
like_number() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
like_number() (*in module chemdataextractor.text*), 141
like_url (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
like_url() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
like_url() (*in module chemdataextractor.text*), 141
list() (*in module chemdataextractor.cli.config*), 35
list() (*in module chemdataextractor.cli.data*), 35
list_of_cems (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
list_of_properties (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
list_of_values (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
ListType (*class in chemdataextractor.model.base*), 58
load() (*chemdataextractor.nlp.tag.ApTagger method*), 87
load() (*chemdataextractor.nlp.tag.AveragedPerceptron method*), 86
load() (*chemdataextractor.nlp.tag.CrfTagger method*), 87
load() (*chemdataextractor.nlp.tag.DictionaryTagger method*), 88
load() (*in module chemdataextractor.cli.cluster*), 35
load_model() (*in module chemdataextractor.data*), 30
local_exists() (*chemdataextractor.data.Package method*), 30
local_path (*chemdataextractor.data.Package attribute*), 30
lower (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
lower() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
lower_count (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
lower_count() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
LStrip (*class in chemdataextractor.text.processors*), 144
LxmlReader (*class in chemdataextractor.reader.markup*), 105

M

make_request() (*chemdataextractor.scrape.base.BaseRequester method*), 130
make_request() (*chemdataextractor.scrape.base.BaseScraper method*), 129
make_request() (*chemdataextractor.scrape.pub.elsevier.ElsevierSearchScraper method*), 125
make_request() (*chemdataextractor.scrape.scraper.GetRequester method*), 137
make_request() (*chemdataextractor.scrape.scraper.PostRequester method*), 137
Mass (*class in chemdataextractor.model.units.mass*), 75
MassModel (*class in chemdataextractor.model.units.mass*), 75
MassUnit (*class in chemdataextractor.model.units.mass*), 75
match() (*in module chemdataextractor.relex.utils*), 114
match_dimensions_of() (*in module chemdataextractor.parse.auto*), 92
match_score() (*in module chemdataextractor.relex.utils*), 114
medpost (*in module chemdataextractor.nlp.corpus*), 82
medpost_evaluation (*in module chemdataextractor.nlp.corpus*), 82
medpost_training (*in module chemdataextractor.nlp.corpus*), 82
MeltingPoint (*class in chemdataextractor.model.model*), 65
memoize() (*in module chemdataextractor.utils*), 31
memoized_property() (*in module chemdataextractor.utils*), 31
merge() (*chemdataextractor.model.model.Compound method*), 63
merge() (*in module chemdataextractor.parse.actions*), 91
merge_all() (*chemdataextractor.model.base.BaseModel method*), 61
merge_contextual() (*chemdataextractor.model.base.BaseModel method*), 61
metadata (*chemdataextractor.biblio.bibtex.BibtexParser attribute*), 32
metadata (*chemdataextractor.doc.document.Document attribute*), 40
MetaData (*class in chemdataextractor.doc.meta*), 43
metadata_author_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
metadata_author_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
metadata_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_date_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
metadata_date_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_doi_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
metadata_doi_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_firstpage_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
metadata_firstpage_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_html_url_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_issue_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
metadata_issue_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_journal_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
metadata_journal_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_language_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_lastpage_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
metadata_lastpage_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_pdf_url_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_pii_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader at-*

tribute), 110
metadata_publisher_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
metadata_publisher_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_title_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
metadata_title_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
metadata_volume_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 110
metadata_volume_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 106
MetaUnit (*class in chemdataextractor.model.units.unit*), 68
Meter (*class in chemdataextractor.model.units.length*), 74
method (*chemdataextractor.model.model.GlassTransition attribute*), 65
Micron (*class in chemdataextractor.model.units.length*), 75
middlename (*chemdataextractor.scrape.pub.springer.SpringerXmlAuthor attribute*), 122
Mile (*class in chemdataextractor.model.units.length*), 74
MINUSES (*in module chemdataextractor.text*), 140
Minute (*class in chemdataextractor.model.units.time*), 77
mode_rows() (*in module chemdataextractor.relex.utils*), 114
model (*chemdataextractor.nlp.cem.CiDictCemTagger attribute*), 81
model (*chemdataextractor.nlp.cem.CrfCemTagger attribute*), 81
model (*chemdataextractor.nlp.cem.CsDictCemTagger attribute*), 81
model (*chemdataextractor.nlp.pos.ApPosTagger attribute*), 85
model (*chemdataextractor.nlp.pos.ChemApPosTagger attribute*), 85
model (*chemdataextractor.nlp.pos.ChemCrfPosTagger attribute*), 85
model (*chemdataextractor.nlp.pos.CrfPosTagger attribute*), 85
model (*chemdataextractor.nlp.tag.DictionaryTagger attribute*), 88
model (*chemdataextractor.nlp.tokenize.ChemSentenceTokenizer attribute*), 89
model (*chemdataextractor.nlp.tokenize.SentenceTokenizer attribute*), 89
model (*chemdataextractor.parse.auto.BaseAutoParser attribute*), 92
model (*chemdataextractor.parse.base.BaseParser attribute*), 93
ModelList (*class in chemdataextractor.model.base*), 62
ModelMeta (*class in chemdataextractor.model.base*), 59
ModelNotFoundError, 31
models (*chemdataextractor.doc.document.Document attribute*), 38
models (*chemdataextractor.doc.element.BaseElement attribute*), 41
models (*chemdataextractor.doc.element.CaptionedElement attribute*), 42
ModelType (*class in chemdataextractor.model.base*), 58
MpParser (*class in chemdataextractor.parse.mp*), 100
multi_entity_phrase_1 (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
multi_entity_phrase_2 (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
multi_entity_phrase_3 (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
multi_entity_phrase_3a (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
multi_entity_phrase_3b (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
multi_entity_phrase_3c (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
multi_entity_phrase_4 (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
multi_entity_phrase_4a (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
multi_entity_phrase_4b (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
multiplicity (*chemdataextractor.model.model.NmrPeak attribute*), 64
MultiQuantityModelTemplateParser (*class in*

chemdataextractor.parse.template), 101

N

name (*chemdataextractor.model.base.BaseType attribute*), 57
name (*chemdataextractor.model.model.Apparatus attribute*), 63
name (*chemdataextractor.scrape.base.BaseField attribute*), 130
name (*chemdataextractor.scrape.pub.rsc.RscLandingSupplement attribute*), 120
name() (*chemdataextractor.scrape.base.BaseScraper method*), 129
NAME_SMALL (*in module chemdataextractor.text*), 140
names (*chemdataextractor.model.model.Compound attribute*), 63
namespaces (*chemdataextractor.scrape.clean.Cleaner attribute*), 132
namespaces (*chemdataextractor.scrape.scraper.RssScraper attribute*), 138
namespaces (*chemdataextractor.scrape.scraper.XmlFormat attribute*), 137
NeelTemperature (*class in chemdataextractor.model.model*), 67
ner_tagged_tokens (*chemdataextractor.doc.text.Sentence attribute*), 54
ner_tagged_tokens (*chemdataextractor.doc.text.Text attribute*), 48
ner_tagger (*chemdataextractor.doc.text.BaseText attribute*), 46
ner_tagger (*chemdataextractor.doc.text.Citation attribute*), 51
ner_tagger (*chemdataextractor.doc.text.Sentence attribute*), 52
ner_tagger (*chemdataextractor.doc.text.Text attribute*), 46
ner_tags (*chemdataextractor.doc.document.Document attribute*), 40
ner_tags (*chemdataextractor.doc.element.CaptionedElement attribute*), 42
ner_tags (*chemdataextractor.doc.text.Sentence attribute*), 54
ner_tags (*chemdataextractor.doc.text.Text attribute*), 48
NlmXmlAuthor (*class in chemdataextractor.scrape.pub.nlm*), 115
NlmXmlDocument (*class in chemdataextractor.scrape.pub.nlm*), 116
NlmXmlImage (*class in chemdataextractor.scrape.pub.nlm*), 115
NlmXmlReader (*class in chemdataextractor.reader.nlm*), 106
NlmXmlTable (*class in chemdataextractor.scrape.pub.nlm*), 116
NmrParser (*class in chemdataextractor.parse.nmr*), 101
NmrPeak (*class in chemdataextractor.model.model*), 64
NmrSpectrum (*class in chemdataextractor.model.model*), 65
NO_SPLIT (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
NO_SPLIT (*chemdataextractor.nlp.tokenize.FineWordTokenizer attribute*), 90
NO_SPLIT (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
NO_SPLIT_CHARS (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
NO_SPLIT_CHEM (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
NO_SPLIT_PREFIX (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
NO_SPLIT_PREFIX (*chemdataextractor.nlp.tokenize.FineWordTokenizer attribute*), 90
NO_SPLIT_PREFIX (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
NO_SPLIT_PREFIX_ENDING (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
NO_SPLIT_SLASH (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
NO_SPLIT_STOP (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
NO_SPLIT_SUFFIX (*chemdataextractor.nlp.tokenize.FineWordTokenizer attribute*), 91
NO_SPLIT_SUFFIX (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
NoMatch (*class in chemdataextractor.parse.elements*), 97
noncontextual_required_fulfilled (*chemdataextractor.model.base.BaseModel attribute*), 60
NoneTagger (*class in chemdataextractor.nlp.tag*), 86

normalize (*in module chemdataextractor.text.normalize*), 143
normalize() (*chemdataextractor.text.normalize.BaseNormalizer method*), 142
normalize() (*chemdataextractor.text.normalize.ChemNormalizer method*), 143
normalize() (*chemdataextractor.text.normalize.ExcessNormalizer method*), 143
normalize() (*chemdataextractor.text.normalize.Normalizer method*), 143
normalized (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
normalized() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
normalizer (*chemdataextractor.nlp.lexicon.ChemLexicon attribute*), 84
normalizer (*chemdataextractor.nlp.lexicon.Lexicon attribute*), 84
Normalizer (*class in chemdataextractor.text.normalize*), 142
Not (*class in chemdataextractor.parse.elements*), 99
nucleus (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
number (*chemdataextractor.model.model.NmrPeak attribute*), 64
NUMBERS (*in module chemdataextractor.text*), 140

O

OneOrMore (*class in chemdataextractor.parse.elements*), 99
online_date (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
online_day (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
online_month (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
online_year (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
Optional (*class in chemdataextractor.parse.elements*), 99
Or (*class in chemdataextractor.parse.elements*), 98
other_solvent (*in module chemdataextractor.parse.cem*), 95

P

Package (*class in chemdataextractor.data*), 30
PACKAGES (*in module chemdataextractor.data*), 30
pages (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
Paragraph (*class in chemdataextractor.doc.text*), 50
paragraphs (*chemdataextractor.doc.document.Document attribute*), 40
paragraphs (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
paragraphs (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
paragraphs (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
params (*chemdataextractor.nlp.cem.CrfCemTagger attribute*), 81
params (*chemdataextractor.nlp.tag.CrfTagger attribute*), 87
parse() (*chemdataextractor.biblio.bibtex.BibtexParser method*), 32
parse() (*chemdataextractor.biblio.xmp.XmpParser method*), 34
parse() (*chemdataextractor.parse.elements.BaseParserElement method*), 97
parse() (*chemdataextractor.reader.base.BaseReader method*), 105
parse() (*chemdataextractor.reader.markup.LxmlReader method*), 106
parse() (*chemdataextractor.reader.pdf.PdfReader method*), 107
parse() (*chemdataextractor.reader.plaintext.PlainTextReader method*), 107
parse_bibtex() (*in module chemdataextractor.biblio.bibtex*), 32
parse_cell() (*chemdataextractor.parse.base.BaseTableParser method*), 94
parse_names() (*chemdataextractor.biblio.bibtex.BibtexParser class method*), 32
parse_rsc_html() (*in module chemdataextractor.scrape.pub.rsc*), 118
parse_sentence() (*chemdataextractor.parse.base.BaseSentenceParser method*), 94
parse_xmp() (*in module chemdataextractor.biblio.xmp*), 34
ParseElementEnhance (*class in chemdataextractor.parse.elements*), 99

ParseException, 96
ParseExpression (*class in chemdataextractor.parse.elements*), 98
parsers (*chemdataextractor.model.base.BaseModel attribute*), 59
parsers (*chemdataextractor.model.model.Apparatus attribute*), 63
parsers (*chemdataextractor.model.model.CNLabel attribute*), 68
parsers (*chemdataextractor.model.model.Compound attribute*), 63
parsers (*chemdataextractor.model.model.CoordinationNumber attribute*), 68
parsers (*chemdataextractor.model.model.CurieTemperature attribute*), 67
parsers (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
parsers (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
parsers (*chemdataextractor.model.model.GlassTransition attribute*), 65
parsers (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
parsers (*chemdataextractor.model.model.IrPeak attribute*), 64
parsers (*chemdataextractor.model.model.IrSpectrum attribute*), 64
parsers (*chemdataextractor.model.model.MeltingPoint attribute*), 65
parsers (*chemdataextractor.model.model.NeelTemperature attribute*), 67
parsers (*chemdataextractor.model.model.NmrPeak attribute*), 65
parsers (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
parsers (*chemdataextractor.model.model.QuantumYield attribute*), 66
parsers (*chemdataextractor.model.model.UvvisPeak attribute*), 63
parsers (*chemdataextractor.model.model.UvvisSpectrum attribute*), 64
parsers (*chemdataextractor.model.units.length.LengthModel attribute*), 74
parsers (*chemdataextractor.model.units.mass.MassModel attribute*), 75
parsers (*chemdataextractor.model.units.quantity_model.DimensionlessModel attribute*), 73
parsers (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 72
parsers (*chemdataextractor.model.units.temperature.TemperatureModel attribute*), 78
parsers (*chemdataextractor.model.units.time.TimeModel attribute*), 76
path (*chemdataextractor.config.Config attribute*), 30
path (*chemdataextractor.scrape.selector.Selector attribute*), 139
Pattern (*class in chemdataextractor.relex.pattern*), 113
patterns (*chemdataextractor.nlp.tag.RegexTagger attribute*), 86
pdf_url (*chemdataextractor.doc.meta.MetaData attribute*), 44
pdf_url (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
pdf_url (*chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute*), 121
pdf_url (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
PdfReader (*class in chemdataextractor.reader.pdf*), 107
peaks (*chemdataextractor.model.model.IrSpectrum attribute*), 64
peaks (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
peaks (*chemdataextractor.model.model.UvvisSpectrum attribute*), 64
perform_search() (*chemdataextractor.scrape.pub.rsc.RscSearchScraper method*), 120
perform_search() (*chemdataextractor.scrape.scraper.SearchScraper method*), 138
PersonName (*class in chemdataextractor.biblio.person*), 32
Phrase (*class in chemdataextractor.relex.phrase*), 113
PlainTextReader (*class in chemdataextractor.reader.plaintext*), 107
PLUSES (*in module chemdataextractor.text*), 140
pmcid (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument at-*

tribute), 116
pmid (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 116
pos_cli() (*in module chemdataextractor.cli.pos*), 37
pos_tagged_tokens (*chemdataextractor.doc.text.Sentence attribute*), 53
pos_tagged_tokens (*chemdataextractor.doc.text.Text attribute*), 47
pos_tagger (*chemdataextractor.doc.text.BaseText attribute*), 46
pos_tagger (*chemdataextractor.doc.text.Sentence attribute*), 52
pos_tagger (*chemdataextractor.doc.text.Text attribute*), 46
pos_tags (*chemdataextractor.doc.text.Sentence attribute*), 54
pos_tags (*chemdataextractor.doc.text.Text attribute*), 47
PostRequester (*class in chemdataextractor.scrape.scraper*), 137
Pound (*class in chemdataextractor.model.units.mass*), 76
predict() (*chemdataextractor.nlp.tag.AveragedPerceptron method*), 86
prefix (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
prefix (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
prepare_gold() (*in module chemdataextractor.cli.chemdner*), 34
prepare_include() (*in module chemdataextractor.cli.dict*), 36
prepare_jochem() (*in module chemdataextractor.cli.dict*), 35
prepare_tokens() (*in module chemdataextractor.cli.chemdner*), 34
PRIMES (*in module chemdataextractor.text*), 140
process() (*chemdataextractor.model.base.BaseType method*), 57
process() (*chemdataextractor.model.base.FloatType method*), 58
process() (*chemdataextractor.model.base.StringType method*), 57
process() (*chemdataextractor.model.units.unit.UnitType method*), 68
process() (*chemdataextractor.scrape.base.BaseField method*), 131
process() (*chemdataextractor.scrape.fields.BoolField method*), 136
process() (*chemdataextractor.scrape.fields.DateTimeField method*), 136
process() (*chemdataextractor.scrape.fields.FloatField method*), 136
process() (*chemdataextractor.scrape.fields.IntField method*), 136
process() (*chemdataextractor.scrape.fields.StringField method*), 135
process() (*chemdataextractor.scrape.fields.UrlField method*), 135
process_abstract (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
process_abstract (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 129
process_abstract (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 118
process_abstract (*chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute*), 121
process_abstract (*chemdataextractor.scrape.pub.rsc.RscLandingDocument attribute*), 120
process_abstract (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
process_abstract (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 125
process_authors (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
process_caption (*chemdataextractor.scrape.pub.elsevier.ElsevierImage attribute*), 126
process_caption (*chemdataextractor.scrape.pub.nlm.NlmXmlImage attribute*), 116
process_caption (*chemdataextractor.scrape.pub.nlm.NlmXmlTable attribute*), 116
process_caption (*chemdataextractor.scrape.pub.rsc.RscImage attribute*), 121
process_caption (*chemdataextractor.scrape.pub.rsc.RscTable attribute*), 121
process_caption (*chemdataextractor.scrape.pub.springer.SpringerXmlImage attribute*), 123
process_caption (*chemdataextractor.scrape.pub.springer.SpringerXmlTable attribute*), 123
process_chemspider_id (*chemdataextrac-*

tor.scrape.pub.rsc.RscChemicalMention attribute), 120
process_doi (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
process_email (*chemdataextractor.scrape.pub.springer.SpringerXmlAuthor attribute*), 122
process_entity() (*chemdataextractor.scrape.base.BaseEntityProcessor method*), 130
process_entity() (*chemdataextractor.scrape.base.BaseScraper method*), 129
process_givennames (*chemdataextractor.scrape.pub.nlm.NlmXmlAuthor attribute*), 115
process_headings (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 125
process_html_url (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
process_html_url (*chemdataextractor.scrape.pub.springer.SpringerHtmlDocument attribute*), 122
process_image_url (*chemdataextractor.scrape.pub.elsevier.ElsevierImage attribute*), 126
process_inchi (*chemdataextractor.scrape.pub.rsc.RscChemicalMention attribute*), 120
process_journal (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
process_landing_url (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
process_lastname (*chemdataextractor.scrape.pub.nlm.NlmXmlAuthor attribute*), 115
process_license (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 125
process_paragraphs (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 125
process_pdf_url (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
process_publisher (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
process_publisher (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 118
process_query() (*chemdataextractor.scrape.scraper.SearchScraper method*), 138
process_response() (*chemdataextractor.scrape.base.BaseFormat method*), 130
process_response() (*chemdataextractor.scrape.base.BaseScraper method*), 129
process_response() (*chemdataextractor.scrape.scraper.HtmlFormat method*), 137
process_response() (*chemdataextractor.scrape.scraper.XmlFormat method*), 137
process_text (*chemdataextractor.scrape.pub.rsc.RscChemicalMention attribute*), 120
process_title (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 135
process_title (*chemdataextractor.scrape.pub.elsevier.ElsevierTable attribute*), 126
process_title (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 118
process_title (*chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute*), 121
process_title (*chemdataextractor.scrape.pub.rsc.RscRssDocument attribute*), 118
process_title (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
process_url() (*chemdataextractor.scrape.scraper.UrlScraper method*), 137
published_date (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
published_date (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
published_date (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 129
published_day (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
published_day (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
published_month (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117

published_month (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
published_year (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
published_year (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
publisher (*chemdataextractor.doc.meta.MetaData attribute*), 43
publisher (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
publisher (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
publisher (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
python_2_unicode_compatible() (*in module chemdataextractor.utils*), 31

Q

quantity_re (*chemdataextractor.doc.text.Sentence attribute*), 54
QUANTITY_RE (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
QuantityModel (*class in chemdataextractor.model.units.quantity_model*), 71
QuantityModelTemplateParser (*class in chemdataextractor.parse.template*), 101
QuantumYield (*class in chemdataextractor.model.model*), 66
QUOTES (*in module chemdataextractor.text*), 140

R

R (*in module chemdataextractor.parse.elements*), 100
RAdd (*class in chemdataextractor.text.processors*), 144
raw_sentences (*chemdataextractor.doc.text.Text attribute*), 47
raw_tokens (*chemdataextractor.doc.text.Sentence attribute*), 53
raw_tokens (*chemdataextractor.doc.text.Text attribute*), 47
raw_units (*chemdataextractor.model.units.quantity_model.DimensionlessModel attribute*), 73
raw_units (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 71
raw_value (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 71
re() (*chemdataextractor.scrape.selector.Selector method*), 139
re() (*chemdataextractor.scrape.selector.SelectorList method*), 139
read() (*chemdataextractor.reader.base.BaseReader method*), 105
read() (*in module chemdataextractor.cli*), 34
ReaderError, 30
readstring() (*chemdataextractor.reader.base.BaseReader method*), 105
received_day (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
received_day (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
received_month (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
received_month (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
received_year (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 117
received_year (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
record_method (*chemdataextractor.model.base.BaseModel attribute*), 62
records (*chemdataextractor.doc.document.BaseDocument attribute*), 38
records (*chemdataextractor.doc.document.Document attribute*), 39
records (*chemdataextractor.doc.element.BaseElement attribute*), 41
records (*chemdataextractor.doc.element.CaptionedElement attribute*), 42
records (*chemdataextractor.doc.figure.Figure attribute*), 42
records (*chemdataextractor.doc.meta.MetaData attribute*), 43
records (*chemdataextractor.doc.table.Table attribute*), 45
records (*chemdataextractor.doc.text.Cell attribute*), 55
records (*chemdataextractor.doc.text.Sentence attribute*), 54
records (*chemdataextractor.doc.text.Text attribute*), 48
records_list (*chemdataextractor.biblio.bibtex.BibtexParser attribute*), 32

reference (*chemdataextractor.scrape.pub.nlm.NlmXmlImage attribute*), 116
reference (*chemdataextractor.scrape.pub.nlm.NlmXmlTable attribute*), 116
reference (*chemdataextractor.scrape.pub.rsc.RscImage attribute*), 121
reference (*chemdataextractor.scrape.pub.rsc.RscTable attribute*), 121
reference (*chemdataextractor.scrape.pub.springer.SpringerXmlImage attribute*), 123
reference (*chemdataextractor.scrape.pub.springer.SpringerXmlTable attribute*), 123
reference_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
reference_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
reference_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
reference_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
reference_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 109
Regex (*class in chemdataextractor.parse.elements*), 98
regex_span_tokenize() (*in module chemdataextractor.nlp.tokenize*), 89
RegexTagger (*class in chemdataextractor.nlp.tag*), 86
registry_number (*in module chemdataextractor.parse.cem*), 95
Relation (*class in chemdataextractor.relex.relationship*), 113
remote_exists() (*chemdataextractor.data.Package method*), 30
remote_path (*chemdataextractor.data.Package attribute*), 30
remove() (*in module chemdataextractor.cli.config*), 35
remove_subsets() (*chemdataextractor.model.base.ModelList method*), 62
replace_rsc_img_chars() (*in module chemdataextractor.scrape.pub.rsc*), 118
required_fields (*chemdataextractor.model.base.ModelMeta attribute*), 59
required_fulfilled (*chemdataextractor.model.base.BaseModel attribute*), 60
reset() (*chemdataextractor.model.base.BaseType method*), 57
reset_updatables() (*chemdataextractor.model.base.BaseModel class method*), 60
reset_vectors() (*chemdataextractor.relex.phrase.Phrase method*), 113
ResponseSearchResult (*class in chemdataextractor.scrape.scraper*), 138
rij_label (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
roles (*chemdataextractor.model.model.Compound attribute*), 63
root (*chemdataextractor.parse.auto.AutoSentenceParser attribute*), 92
root (*chemdataextractor.parse.auto.AutoTableParser attribute*), 92
root (*chemdataextractor.parse.base.BaseParser attribute*), 93
root (*chemdataextractor.parse.cem.ChemicalLabelParser attribute*), 95
root (*chemdataextractor.parse.cem.CompoundHeadingParser attribute*), 95
root (*chemdataextractor.parse.cem.CompoundParser attribute*), 95
root (*chemdataextractor.parse.cem.CompoundTableParser attribute*), 95
root (*chemdataextractor.parse.ir.IrParser attribute*), 100
root (*chemdataextractor.parse.mp.MpParser attribute*), 100
root (*chemdataextractor.parse.nmr.NmrParser attribute*), 101
root (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 103
root (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
root (*chemdataextractor.parse.tg.TgParser attribute*), 103
root (*chemdataextractor.parse.uvvis.UvvisParser attribute*), 103
root (*chemdataextractor.scrape.base.BaseScraper attribute*), 129
root (*chemdataextractor.scrape.pub.rsc.RscSearchScraper attribute*), 119
root (*chemdataextractor.scrape.scraper.RssScraper attribute*), 138
root_css (*chemdataextrac-*

tor.reader.acs.AcsHtmlReader attribute), 104
root_css (*chemdataextractor.reader.cssp.CsspHtmlReader attribute*), 105
root_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
root_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
root_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
root_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
root_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
root_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
root_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 108
root_xpath (*chemdataextractor.scrape.base.BaseScraper attribute*), 129
rows (*chemdataextractor.scrape.pub.elsevier.ElsevierTableData attribute*), 126
rows (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlTableData attribute*), 127
rsc_html_whitespace() (*in module chemdataextractor.reader.rsc*), 108
RSC_IMG_CHARS (*in module chemdataextractor.scrape.pub.rsc*), 118
rsc_substitute (*in module chemdataextractor.scrape.pub.rsc*), 118
RscChemicalMention (*class in chemdataextractor.scrape.pub.rsc*), 120
RscHtmlDocument (*class in chemdataextractor.scrape.pub.rsc*), 121
RscHtmlReader (*class in chemdataextractor.reader.rsc*), 108
RscHtmlScraper (*class in chemdataextractor.scrape.pub.rsc*), 122
RscImage (*class in chemdataextractor.scrape.pub.rsc*), 120
RscLandingDocument (*class in chemdataextractor.scrape.pub.rsc*), 120
RscLandingScraper (*class in chemdataextractor.scrape.pub.rsc*), 120
RscLandingSupplement (*class in chemdataextractor.scrape.pub.rsc*), 120
RscRssDocument (*class in chemdataextractor.scrape.pub.rsc*), 118
RscRssScraper (*class in chemdataextractor.scrape.pub.rsc*), 119
RscSearchDocument (*class in chemdataextractor.scrape.pub.rsc*), 119
RscSearchScraper (*class in chemdataextractor.scrape.pub.rsc*), 119
RscTable (*class in chemdataextractor.scrape.pub.rsc*), 121
RssScraper (*class in chemdataextractor.scrape.scraper*), 138
RStrip (*class in chemdataextractor.text.processors*), 144
run() (*chemdataextractor.scrape.pub.elsevier.ElsevierSearchScraper method*), 125
run() (*chemdataextractor.scrape.scraper.SearchScraper method*), 138
run() (*chemdataextractor.scrape.scraper.UrlScraper method*), 138
run() (*in module chemdataextractor.cli.evaluate*), 36

S

safe_name() (*in module chemdataextractor.parse.elements*), 96
save() (*chemdataextractor.nlp.tag.ApTagger method*), 87
save() (*chemdataextractor.nlp.tag.AveragedPerceptron method*), 86
save() (*chemdataextractor.nlp.tag.DictionaryTagger method*), 88
scan() (*chemdataextractor.parse.elements.BaseParserElement method*), 96
schemes (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
scrape() (*chemdataextractor.scrape.base.BaseField method*), 131
scrape() (*chemdataextractor.scrape.entity.Entity class method*), 133
scrape() (*chemdataextractor.scrape.fields.EntityField method*), 136
SearchResult (*class in chemdataextractor.scrape.scraper*), 138
SearchScraper (*class in chemdataextractor.scrape.scraper*), 138
Second (*class in chemdataextractor.model.units.time*), 77

selector (*chemdataextractor.scrape.scraper.ResponseSearchResult attribute*), 138
selector (*chemdataextractor.scrape.scraper.SearchResult attribute*), 138
selector (*chemdataextractor.scrape.scraper.SeleniumSearchResult attribute*), 138
Selector (*class in chemdataextractor.scrape.selector*), 139
SelectorList (*class in chemdataextractor.scrape.selector*), 139
SeleniumSearchResult (*class in chemdataextractor.scrape.scraper*), 138
Sentence (*class in chemdataextractor.doc.text*), 52
sentence_tokenizer (*chemdataextractor.doc.text.Text attribute*), 47
sentences (*chemdataextractor.doc.text.Text attribute*), 47
sentences() (*in module chemdataextractor.cli.tokenize*), 37
SentenceTokenizer (*class in chemdataextractor.nlp.tokenize*), 89
serialize() (*chemdataextractor.doc.document.Document method*), 40
serialize() (*chemdataextractor.doc.element.CaptionedElement method*), 42
serialize() (*chemdataextractor.doc.meta.MetaData method*), 43
serialize() (*chemdataextractor.doc.table.Table method*), 45
serialize() (*chemdataextractor.doc.text.BaseText method*), 46
serialize() (*chemdataextractor.model.base.BaseModel method*), 60
serialize() (*chemdataextractor.model.base.BaseType method*), 57
serialize() (*chemdataextractor.model.base.ListType method*), 59
serialize() (*chemdataextractor.model.base.ModelList method*), 62
serialize() (*chemdataextractor.model.base.ModelType method*), 58
serialize() (*chemdataextractor.model.base.SetType method*), 59
serialize() (*chemdataextractor.model.units.unit.UnitType method*), 68
serialize() (*chemdataextractor.relex.entity.Entity method*), 112
serialize() (*chemdataextractor.relex.relationship.Relation method*), 114
serialize() (*chemdataextractor.scrape.base.BaseField method*), 131
serialize() (*chemdataextractor.scrape.entity.Entity method*), 133
serialize() (*chemdataextractor.scrape.entity.EntityList method*), 134
serialize() (*chemdataextractor.scrape.fields.DateTimeField method*), 136
set() (*in module chemdataextractor.cli.config*), 35
set_action() (*chemdataextractor.parse.elements.BaseParserElement method*), 96
set_config() (*chemdataextractor.doc.text.Text method*), 47
set_name() (*chemdataextractor.parse.elements.BaseParserElement method*), 96
SetType (*class in chemdataextractor.model.base*), 59
shape (*chemdataextractor.model.model.UvvisPeak attribute*), 63
shape (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
shape() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
shift (*chemdataextractor.model.model.NmrPeak attribute*), 64
single_cem (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
SINGLE_QUOTES (*in module chemdataextractor.text*), 140
single_specifier_and_value (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
single_specifier_and_value_with_optional_unit (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
Singleton (*class in chemdataextractor.utils*), 31
size (*chemdataextractor.biblio.bibtex.BibtexParser attribute*), 32
SkipTo (*class in chemdataextractor.parse.elements*), 99
SLASHES (*in module chemdataextractor.text*), 140
SMALL (*in module chemdataextractor.text*), 140
solvent (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
solvent (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
solvent (*chemdataextractor.model.model.IrSpectrum attribute*), 64
solvent (*chemdataextractor.model.model.MeltingPoint attribute*), 65

solvent (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
solvent (*chemdataextractor.model.model.QuantumYield attribute*), 66
solvent (*chemdataextractor.model.model.UvvisSpectrum attribute*), 63
space_labels() (*in module chemdataextractor.scrape.pub.nlm*), 115
space_references() (*in module chemdataextractor.scrape.pub.rsc*), 118
Span (*class in chemdataextractor.doc.text*), 55
span_tokenize() (*chemdataextractor.nlp.tokenize.BaseTokenizer method*), 88
span_tokenize() (*chemdataextractor.nlp.tokenize.SentenceTokenizer method*), 89
span_tokenize() (*chemdataextractor.nlp.tokenize.WordTokenizer method*), 90
species (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
specifier (*chemdataextractor.model.base.BaseModel attribute*), 59
specifier (*chemdataextractor.model.model.CNLabel attribute*), 68
specifier (*chemdataextractor.model.model.CoordinationNumber attribute*), 67
specifier (*chemdataextractor.model.model.CurieTemperature attribute*), 67
specifier (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
specifier (*chemdataextractor.model.model.NeelTemperature attribute*), 67
specifier (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 72
specifier_and_value (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
specifier_before_cem_and_value_phrase (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
specifier_expression (*chemdataextractor.model.model.CoordinationNumber attribute*), 67
specifier_expression (*chemdataextractor.model.model.InteratomicDistance attribute*), 67
specifier_phrase (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
specifier_phrase (*chemdataextractor.parse.template.QuantityModelTemplateParser attribute*), 101
SPLIT (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
SPLIT (*chemdataextractor.nlp.tokenize.FineWordTokenizer attribute*), 90
SPLIT (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
SPLIT_END (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
SPLIT_END_NO_DIGIT (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
SPLIT_END_WORD (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
split_last_stop (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 90
SPLIT_NO_DIGIT (*chemdataextractor.nlp.tokenize.FineWordTokenizer attribute*), 90
SPLIT_NO_DIGIT (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
SPLIT_START_WORD (*chemdataextractor.nlp.tokenize.WordTokenizer attribute*), 89
SPLIT_SUFFIX (*chemdataextractor.nlp.tokenize.ChemWordTokenizer attribute*), 90
SPLITS (*in module chemdataextractor.nlp.cem*), 81
springer_html_whitespace() (*in module chemdataextractor.reader.springer*), 110
SpringerHtmlDocument (*class in chemdataextractor.scrape.pub.springer*), 122
SpringerHtmlReader (*class in chemdataextractor.reader.springer*), 110
SpringerMaterialsHtmlReader (*class in chemdataextractor.reader.springer*), 110
SpringerXmlAuthor (*class in chemdataextractor.scrape.pub.springer*), 122
SpringerXmlDocument (*class in chemdataextrac-*

tor.scrape.pub.springer), 123
SpringerXmlImage (*class in chemdataextractor.scrape.pub.springer*), 123
SpringerXmlTable (*class in chemdataextractor.scrape.pub.springer*), 123
src (*chemdataextractor.scrape.pub.nlm.NlmXmlTable attribute*), 116
src (*chemdataextractor.scrape.pub.rsc.RscTable attribute*), 121
src (*chemdataextractor.scrape.pub.springer.SpringerXmlTable attribute*), 123
standard (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
standard (*chemdataextractor.model.model.QuantumYield attribute*), 66
standard_solvent (*chemdataextractor.model.model.QuantumYield attribute*), 66
standard_units (*chemdataextractor.model.units.dimension.Dimension attribute*), 71
standard_units (*chemdataextractor.model.units.dimension.Dimensionless attribute*), 71
standard_units (*chemdataextractor.model.units.length.Length attribute*), 73
standard_units (*chemdataextractor.model.units.mass.Mass attribute*), 75
standard_units (*chemdataextractor.model.units.temperature.Temperature attribute*), 78
standard_units (*chemdataextractor.model.units.time.Time attribute*), 76
standard_units (*in module chemdataextractor.model.units*), 68
standard_units (*in module chemdataextractor.model.units.dimension*), 70
standard_value (*chemdataextractor.model.model.QuantumYield attribute*), 66
standardize_role() (*in module chemdataextractor.parse.cem*), 95
start (*chemdataextractor.doc.text.Sentence attribute*), 53
start (*chemdataextractor.doc.text.Span attribute*), 56
START (*chemdataextractor.nlp.tag.ApTagger attribute*), 87
Start (*class in chemdataextractor.parse.elements*), 98
STOP_RES (*in module chemdataextractor.nlp.cem*), 80
STOP_SUB (*in module chemdataextractor.nlp.cem*), 80
STOP_TOKENS (*in module chemdataextractor.nlp.cem*), 80
STOPLIST (*in module chemdataextractor.nlp.cem*), 80
streamline() (*chemdataextractor.parse.elements.BaseParserElement method*), 97
streamline() (*chemdataextractor.parse.elements.ParseElementEnhance method*), 99
streamline() (*chemdataextractor.parse.elements.ParseExpression method*), 98
streamlined (*chemdataextractor.parse.elements.BaseParserElement attribute*), 96
strength (*chemdataextractor.model.model.IrPeak attribute*), 64
strict_chemical_label (*in module chemdataextractor.parse.cem*), 94
strict_normalize (*in module chemdataextractor.text.normalize*), 143
StringField (*class in chemdataextractor.scrape.fields*), 135
StringType (*class in chemdataextractor.model.base*), 57
strip (*in module chemdataextractor.scrape.clean*), 132
strip_cit_html (*in module chemdataextractor.scrape.pub.rsc*), 118
strip_delta() (*in module chemdataextractor.parse.nmr*), 101
STRIP_END (*in module chemdataextractor.nlp.cem*), 80
strip_html (*in module chemdataextractor.scrape.clean*), 132
strip_markup (*in module chemdataextractor.scrape.clean*), 132
strip_pmc_abstract_xml (*in module chemdataextractor.scrape.pub.nlm*), 115
strip_pmc_paragraph_xml (*in module chemdataextractor.scrape.pub.nlm*), 115
strip_pmc_xml (*in module chemdataextractor.scrape.pub.nlm*), 115
strip_querystring() (*in module chemdataextractor.text.processors*), 144
strip_rsc_html (*in module chemdataextractor.scrape.pub.rsc*), 118
strip_springer_abstract_xml (*in module chemdataextractor.scrape.pub.springer*), 122
strip_springer_xml (*in module chemdataextractor.scrape.pub.springer*), 122
STRIP_START (*in module chemdataextractor.nlp.cem*), 80
strip_stop() (*in module chemdataextractor.parse.actions*), 91
strip_xpath (*chemdataextractor.scrape.clean.Cleaner attribute*), 131

sub_headings (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
subfinder() (*in module chemdataextractor.relex.utils*), 114
Substitutor (*class in chemdataextractor.text.processors*), 144
suffix (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 83
suffix (*chemdataextractor.scrape.pub.springer.SpringerXmlAuthor attribute*), 122
suffix() (*chemdataextractor.nlp.lexicon.Lexicon method*), 84
supplements (*chemdataextractor.scrape.pub.rsc.RscLandingDocument attribute*), 120

T

T (*in module chemdataextractor.parse.elements*), 100
Table (*class in chemdataextractor.doc.table*), 44
table_body_row_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
table_body_row_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
table_body_row_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
table_body_row_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
table_body_row_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
table_body_row_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
table_body_row_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 109
table_caption_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
table_caption_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
table_caption_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
table_caption_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
table_caption_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
table_caption_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
table_caption_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
table_cell_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
table_cell_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
table_cell_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
table_cell_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
table_cell_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 109
table_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
table_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
table_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
table_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
table_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
table_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
table_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
table_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 108
table_footnote_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
table_footnote_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109

table_footnote_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
table_footnote_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
table_footnote_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
table_head_row_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
table_head_row_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
table_head_row_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
table_head_row_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
table_head_row_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
table_head_row_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
tables (*chemdataextractor.doc.document.Document attribute*), 39
tables (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 127
tables (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 129
tables (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
tag (*chemdataextractor.scrape.selector.Selector attribute*), 139
Tag (*class in chemdataextractor.parse.elements*), 97
tag() (*chemdataextractor.nlp.cem.CemTagger method*), 81
tag() (*chemdataextractor.nlp.tag.ApTagger method*), 87
tag() (*chemdataextractor.nlp.tag.BaseTagger method*), 85
tag() (*chemdataextractor.nlp.tag.CrfTagger method*), 87
tag() (*chemdataextractor.nlp.tag.DictionaryTagger method*), 88
tag() (*chemdataextractor.nlp.tag.NoneTagger method*), 86
tag() (*chemdataextractor.nlp.tag.RegexTagger method*), 86
tag() (*in module chemdataextractor.cli.chemdner*), 34
tag() (*in module chemdataextractor.cli.dict*), 36
tag() (*in module chemdataextractor.cli.pos*), 37
tag_sents() (*chemdataextractor.nlp.tag.BaseTagger method*), 85
tagged_tokens (*chemdataextractor.doc.text.Sentence attribute*), 54
tagged_tokens (*chemdataextractor.doc.text.Text attribute*), 48
taggers (*chemdataextractor.nlp.cem.CemTagger attribute*), 81
tags (*chemdataextractor.doc.text.BaseText attribute*), 46
tags (*chemdataextractor.doc.text.Sentence attribute*), 54
tags (*chemdataextractor.doc.text.Text attribute*), 48
TAGS (*in module chemdataextractor.nlp.pos*), 85
tde_table (*chemdataextractor.doc.table.Table attribute*), 45
temperature (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
temperature (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
temperature (*chemdataextractor.model.model.IrSpectrum attribute*), 64
temperature (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
temperature (*chemdataextractor.model.model.QuantumYield attribute*), 66
temperature (*chemdataextractor.model.model.UvvisSpectrum attribute*), 63
Temperature (*class in chemdataextractor.model.units.temperature*), 78
temperature_units (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 67
temperature_units (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
temperature_units (*chemdataextractor.model.model.IrSpectrum attribute*), 64
temperature_units (*chemdataextractor.model.model.NmrSpectrum attribute*), 65
temperature_units (*chemdataextractor.model.model.QuantumYield attribute*), 66
temperature_units (*chemdataextractor.model.model.UvvisSpectrum attribute*), 64

TemperatureModel (*class in chemdataextractor.model.units.temperature*), 78
TemperatureUnit (*class in chemdataextractor.model.units.temperature*), 78
test (*chemdataextractor.scrape.pub.elsevier.ElsevierSearchDocument attribute*), 125
text (*chemdataextractor.doc.text.BaseText attribute*), 46
text (*chemdataextractor.doc.text.Span attribute*), 56
text (*chemdataextractor.nlp.lexicon.Lexeme attribute*), 82
text (*chemdataextractor.scrape.pub.rsc.RscChemicalMention attribute*), 120
Text (*class in chemdataextractor.doc.text*), 46
textnode (*chemdataextractor.scrape.csstranslator.CdeXPathExpr attribute*), 133
TgParser (*class in chemdataextractor.parse.tg*), 103
tidy_nlm_references() (*in module chemdataextractor.scrape.pub.nlm*), 115
tidy_springer_references() (*in module chemdataextractor.scrape.pub.springer*), 122
TILDES (*in module chemdataextractor.text*), 140
Time (*class in chemdataextractor.model.units.time*), 76
TimeModel (*class in chemdataextractor.model.units.time*), 76
TimeUnit (*class in chemdataextractor.model.units.time*), 76
title (*chemdataextractor.doc.meta.MetaData attribute*), 43
title (*chemdataextractor.scrape.entity.DocumentEntity attribute*), 134
title (*chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute*), 126
title (*chemdataextractor.scrape.pub.elsevier.ElsevierTable attribute*), 126
title (*chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute*), 128
title (*chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute*), 116
title (*chemdataextractor.scrape.pub.rsc.RscHtmlDocument attribute*), 121
title (*chemdataextractor.scrape.pub.rsc.RscRssDocument attribute*), 118
title (*chemdataextractor.scrape.pub.rsc.RscSearchDocument attribute*), 119
title (*chemdataextractor.scrape.pub.springer.SpringerHtmlDocument attribute*), 122
title (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 123
Title (*class in chemdataextractor.doc.text*), 48
title_css (*chemdataextractor.reader.acs.AcsHtmlReader attribute*), 104
title_css (*chemdataextractor.reader.cssp.CsspHtmlReader attribute*), 105
title_css (*chemdataextractor.reader.elsevier.ElsevierXmlReader attribute*), 109
title_css (*chemdataextractor.reader.markup.LxmlReader attribute*), 105
title_css (*chemdataextractor.reader.nlm.NlmXmlReader attribute*), 107
title_css (*chemdataextractor.reader.rsc.RscHtmlReader attribute*), 108
title_css (*chemdataextractor.reader.springer.SpringerHtmlReader attribute*), 111
title_css (*chemdataextractor.reader.springer.SpringerMaterialsHtmlReader attribute*), 110
title_css (*chemdataextractor.reader.uspto.UsptoXmlReader attribute*), 108
titles (*chemdataextractor.doc.document.Document attribute*), 40
to_json() (*chemdataextractor.doc.document.Document method*), 40
to_json() (*chemdataextractor.doc.element.BaseElement method*), 41
to_json() (*chemdataextractor.model.base.BaseModel method*), 60
to_json() (*chemdataextractor.model.base.ModelList method*), 62
to_json() (*chemdataextractor.scrape.entity.Entity method*), 134
to_json() (*chemdataextractor.scrape.entity.EntityList method*), 134
to_string() (*chemdataextractor.relex.pattern.Pattern method*), 113
to_string() (*chemdataextractor.relex.phrase.Phrase method*), 113
Token (*class in chemdataextractor.doc.text*), 56
tokenize() (*chemdataextrac-*

tor.nlp.tokenize.BaseTokenizer method), 88
tokenize_cli() (*in module chemdataextractor.cli.tokenize*), 37
tokens (*chemdataextractor.doc.text.BaseText attribute*), 46
tokens (*chemdataextractor.doc.text.Sentence attribute*), 53
tokens (*chemdataextractor.doc.text.Text attribute*), 47
Tonne (*class in chemdataextractor.model.units.mass*), 76
train() (*chemdataextractor.nlp.tag.ApTagger method*), 87
train() (*chemdataextractor.nlp.tag.CrfTagger method*), 87
train() (*in module chemdataextractor.cli.pos*), 37
train_all() (*in module chemdataextractor.cli.pos*), 37
train_crf() (*in module chemdataextractor.cli.cem*), 34
train_perceptron() (*in module chemdataextractor.cli.pos*), 37
train_punkt() (*in module chemdataextractor.cli.tokenize*), 37
TranslatorMixin (*class in chemdataextractor.scrape.csstranslator*), 133
treebank2_development (*in module chemdataextractor.nlp.corpus*), 82
treebank2_evaluation (*in module chemdataextractor.nlp.corpus*), 82
treebank2_training (*in module chemdataextractor.nlp.corpus*), 82
trigger_phrase (*chemdataextractor.parse.auto.AutoSentenceParser attribute*), 92
trigger_phrase (*chemdataextractor.parse.base.BaseParser attribute*), 93
try_parse() (*chemdataextractor.parse.elements.BaseParserElement method*), 97
type (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 66
type (*chemdataextractor.model.model.QuantumYield attribute*), 66

U

ui (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 123
unapostrophe() (*in module chemdataextractor.text.processors*), 145
unit (*chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute*), 102
Unit (*class in chemdataextractor.model.units.unit*), 68
units (*chemdataextractor.model.model.ElectrochemicalPotential attribute*), 66
units (*chemdataextractor.model.model.FluorescenceLifetime attribute*), 66
units (*chemdataextractor.model.model.GlassTransition attribute*), 65
units (*chemdataextractor.model.model.IrPeak attribute*), 64
units (*chemdataextractor.model.model.QuantumYield attribute*), 66
units (*chemdataextractor.model.model.UvvisPeak attribute*), 63
units (*chemdataextractor.model.units.quantity_model.QuantityModel attribute*), 71
units_dict (*chemdataextractor.model.units.dimension.Dimension attribute*), 70
units_dict (*chemdataextractor.model.units.length.Length attribute*), 73
units_dict (*chemdataextractor.model.units.mass.Mass attribute*), 75
units_dict (*chemdataextractor.model.units.temperature.Temperature attribute*), 78
units_dict (*chemdataextractor.model.units.time.Time attribute*), 76
UnitType (*class in chemdataextractor.model.units.unit*), 68
unprocessed_ner_tagged_tokens (*chemdataextractor.doc.text.Sentence attribute*), 54
unprocessed_ner_tagged_tokens (*chemdataextractor.doc.text.Text attribute*), 48
unprocessed_ner_tags (*chemdataextractor.doc.text.Sentence attribute*), 54
unprocessed_ner_tags (*chemdataextractor.doc.text.Text attribute*), 48
update() (*chemdataextractor.model.base.BaseModel class method*), 60
update() (*chemdataextractor.model.model.Compound class method*), 63
update() (*chemdataextractor.nlp.tag.AveragedPerceptron method*), 86
update_dictionaries() (*chemdataextractor.relex.cluster.Cluster method*), 111
update_pattern() (*chemdataextractor.relex.cluster.Cluster method*), 112
update_pattern_confidence() (*chemdataex-*

tractor.relex.cluster.Cluster method), 112
update_weights() (chemdataextractor.relex.cluster.Cluster method), 112
updated (chemdataextractor.model.base.BaseModel attribute), 60
upper_count (chemdataextractor.nlp.lexicon.Lexeme attribute), 83
upper_count() (chemdataextractor.nlp.lexicon.Lexicon method), 84
url (chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute), 128
url (chemdataextractor.scrape.pub.rsc.RscImage attribute), 120
url (chemdataextractor.scrape.pub.rsc.RscLandingSupplement attribute), 120
url_prefix (chemdataextractor.reader.elsevier.ElsevierXmlReader attribute), 110
UrlField (class in chemdataextractor.scrape.fields), 135
UrlScraper (class in chemdataextractor.scrape.scraper), 137
UsptoXmlReader (class in chemdataextractor.reader.uspto), 108
UvvisParser (class in chemdataextractor.parse.uvvis), 103
UvvisPeak (class in chemdataextractor.model.model), 63
UvvisSpectrum (class in chemdataextractor.model.model), 63

V

value (chemdataextractor.model.model.ElectrochemicalPotential attribute), 66
value (chemdataextractor.model.model.FluorescenceLifetime attribute), 66
value (chemdataextractor.model.model.GlassTransition attribute), 65
value (chemdataextractor.model.model.IrPeak attribute), 64
value (chemdataextractor.model.model.QuantumYield attribute), 66
value (chemdataextractor.model.model.UvvisPeak attribute), 63
value (chemdataextractor.model.units.quantity_model.QuantityModel attribute), 71
value_phrase (chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute), 102
value_phrase (chemdataextractor.parse.template.QuantityModelTemplateParser attribute), 101
value_specifier_cem_phrase (chemdataextractor.parse.template.QuantityModelTemplateParser attribute), 101
value_with_optional_unit (chemdataextractor.parse.template.MultiQuantityModelTemplateParser attribute), 102
values() (chemdataextractor.model.base.BaseModel method), 60
verbose() (in module chemdataextractor.relex.utils), 114
volume (chemdataextractor.doc.meta.MetaData attribute), 43
volume (chemdataextractor.scrape.entity.DocumentEntity attribute), 134
volume (chemdataextractor.scrape.pub.elsevier.ElsevierHtmlDocument attribute), 127
volume (chemdataextractor.scrape.pub.elsevier.ElsevierXmlDocument attribute), 128
volume (chemdataextractor.scrape.pub.nlm.NlmXmlDocument attribute), 117
volume (chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute), 124

W

W (in module chemdataextractor.parse.elements), 100
where() (in module chemdataextractor.cli.data), 35
with_condition() (chemdataextractor.parse.elements.BaseParserElement method), 96
Word (class in chemdataextractor.parse.elements), 97
word_shape() (in module chemdataextractor.text), 141
word_tokenizer (chemdataextractor.doc.text.BaseText attribute), 46
word_tokenizer (chemdataextractor.doc.text.Sentence attribute), 52
word_tokenizer (chemdataextractor.doc.text.Text attribute), 46
words() (in module chemdataextractor.cli.tokenize), 37
WordTokenizer (class in chemdataextractor.nlp.tokenize), 89
wrap() (chemdataextractor.parse.elements.ParseException class method), 96
wsj (in module chemdataextractor.nlp.corpus), 82
wsj_development (in module chemdataextractor.nlp.corpus), 82

wsj_evaluation (*in module chemdataextractor.nlp.corpus*), 82
wsj_training (*in module chemdataextractor.nlp.corpus*), 82

X

XmlFormat (*class in chemdataextractor.scrape.scraper*), 137
XmlReader (*class in chemdataextractor.reader.markup*), 106
XmpParser (*class in chemdataextractor.biblio.xmp*), 33
xpath() (*chemdataextractor.scrape.selector.Selector method*), 139
xpath() (*chemdataextractor.scrape.selector.SelectorList method*), 139
xpath_attr_functional_pseudo_element() (*chemdataextractor.scrape.csstranslator.TranslatorMixin method*), 133
xpath_element() (*chemdataextractor.scrape.csstranslator.TranslatorMixin method*), 133
xpath_pseudo_element() (*chemdataextractor.scrape.csstranslator.TranslatorMixin method*), 133
xpath_text_simple_pseudo_element() (*chemdataextractor.scrape.csstranslator.TranslatorMixin method*), 133

Y

year (*chemdataextractor.scrape.pub.springer.SpringerXmlDocument attribute*), 124
Year (*class in chemdataextractor.model.units.time*), 77

Z

ZeroOrMore (*class in chemdataextractor.parse.elements*), 99